Untersuchung des Parallelisierungspotentials diskreter ereignisgesteuerter Simulationen am Beispiel der Schaltkreissimulation

Dem Fachbereich Informatik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs (Dr.-Ing.)
vorgelegte

Dissertation

von
Diplom-Informatiker

Gerd Meister

aus Kaiserslautern

Referent: Prof. Dr. F. Mattern
Korreferent: Prof. Dr. P. Kammerer
Tag der Einreichung: 9. August 1999
Tag der mündlichen Prüfung: 14. Oktober 1999

Darmstadt 1999
D 17
Darmstädter Dissertation

Untersuchung des Parallelisierungspotentials diskreter ereignisgesteuerter Simulationen am Beispiel der Schaltkreissimulation

Dem Fachbereich Informatik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs (Dr.-Ing.)
vorgelegte

Dissertation

von
Diplom-Informatiker

Gerd Meister

aus Kaiserslautern

Referent: Prof. Dr. F. Mattern
Korreferent: Prof. Dr. P. Kammerer
Tag der Einreichung: 9. August 1999
Tag der mündlichen Prüfung: 14. Oktober 1999

Darmstadt 1999
D 17
Darmstädter Dissertation

Befiehl dem HERRN deine Wege und hoffe auf IHN,
er wird's wohl machen.
Psalm 37, 5.

Danksagungen

Ich bedanke mich an dieser Stelle bei allen, die mich bei der Anfertigung dieser Arbeit unterstützt haben. An erster Stelle steht mein Doktorvater, Prof. Dr. Friedemann Mattern, der durch seine freizügige Art und Motivation viel Spielraum zum Erforschen neuer, interessanter Aspekte eröffnete. Herrn Prof. Dr. Peter Kammerer danke herzlich ich für die Übernahme des Korreferats.

Allen meinen Kolleginnen und Kollegen danke ich für die gute Atmosphäre am Fachgebiet Verteilte Systeme und besonders Gerd Aschemann für die tiefen Einblicke (oder sind es eher Abgründe?) in die Systemverwaltung, Stefan Fünfrocken als langjährigem Zimmergenossen und Marion Franz für die nie endenden Ermutigungen.

Bei meiner Frau Simone möchte ich mich bedanken, daß sie mich immer wieder daran erinnert hat, daß es neben der Informatik auch noch andere Dinge im Leben gibt. Die vielen Radtouren holten mich oft wieder auf den Boden der Realität zurück.

Für die Nutzung des Parallelrechners in Paderborn gilt mein Dank Bernhard Bauer vom PC^2, der mich in den erlesenen Kreis der Nutzer aufnahm, und Alex Keller sowie den anderen Gurus, die bei Problemen stets kompetente Ansprechpartner waren.

Dank sei auch der Deutschen Bahn, die mir durch stetig schlechter werdende Verbindungen zwischen Kaiserslautern und Darmstadt immer mehr Freiräume zum Literaturstudium schuf.

Zuguterletzt, aber nicht an letzter Stelle, danke ich meinen Eltern für ihre Liebe und Toleranz, mit der sie mir einen guten Weg im Leben vorgebahnt haben und mir dennoch die Freiheit für eigene Schritte ließen.

Darmstadt, im Juli 1999

Gerd Meister

Zusammenfassung

Mit dem Aufkommen leistungsfähiger Parallelrechner vor ca. 20 Jahren stellte sich in vielen Anwendungsbereichen die Frage, ob der Einsatz dieser Rechner eine wesentliche Reduzierung der Gesamtlaufzeit gegebener Applikationen ermöglicht und ob nunmehr auch große und größte Probleme in vertretbaren Zeiten berechenbar sind. Als wichtige Applikationsklasse wurden seit Ende der siebziger Jahre *Simulationen* hinsichtlich des Beschleunigungspotentials untersucht [BRY79a, CHM81a, JEF85a, MIS86a].

Neben dem Bereich der kontinuierlichen Simulation, die vor allem für die naturwissenschaftlichen Disziplinen interessant ist und meist auf datenparallelen Ansätzen basiert, stellt diesbezüglich die Simulation *zeitdiskreter Systeme* aufgrund des heterogenen Aufbaus vieler Simulationsmodelle aus den Bereichen der technischen Wissenschaften die für die Informatik-Forschung weitaus interessantere Variante dar. Einsatzgebiete der zeitdiskreten Methodik sind Simulationen, bei denen die dem Modell zugrundeliegenden Systeme auf disjunkte Objekte und Interaktionen zwischen den Objekten abbildbar sind.

Diese Simulationsobjekte werden jeweils durch eine Menge von Zustandsvariablen repräsentiert. Die Objekte erfahren zu diskreten Zeitpunkten aufgrund atomar eintretender Ereignisse Zustandsänderungen. Als Reaktion darauf werden die Zustandsvariablen modifiziert und neue Ereignisse, die durchaus auch andere Objekte betreffen können, zu einem späteren Zeitpunkt eingeplant. Da zwischen zwei Ereignissen keine Zustandsänderungen eintreten, werden Totzeiten durch Weiterschalten der Simulationszeit auf den Zeitpunkt des nächsten Ereignisses übersprungen. Daher resultiert auch der Name *diskrete ereignisgesteuerte Simulation (DES)*. Im Gegensatz dazu schreitet bei der zeitgesteuerten Simulation die Zeit in äquidistanten Abschnitten fort.

Bei der *Parallelisierung* der DES werden die Objekte des Modells in disjunkte *Teilmodelle* unterteilt. Jedes Teilmodell wird autonom von einem eigenen Simulator bearbeitet, indem stets das Ereignis mit dem für diesen Simulator kleinsten Zeitstempel ausgeführt wird. Er operiert auf seinen lokalen Objekten prinzipiell ebenso wie ein sequentieller Simulator auf dem Gesamtmodell. Analog zum nicht verteilten Fall wird ein lokales Ereignis ausgeführt, indem die Zustandsänderungen des betroffenen Objekts vollzogen und neue Ereignisse generiert werden.

Aufgrund der Verteilung des Modells erzeugen die Simulatoren auch Ereignisse für nicht-lokale Objekte. Durch das Überspringen von Totzeiten schreiten die Uhren der sequentiellen Teilsimulatoren mit unterschiedlicher Geschwindigkeit fort. Wird nun ein Ereignis einem anderen Simulator eingeplant, so muß stets garantiert werden, daß der Empfänger dieses auch in der korrekten zeitlichen Reihenfolge ausführt. Die Simulationszeit des Zielsimulators kann durch das asynchrone Verhalten der Uhren bereits höher sein als der geplante Ausführungszeitpunkt des neuen Ereignisses. Die Gewährleistung der korrekten Ereignisreihenfolge stellt eines der Grundprobleme der *parallelen diskreten ereignisgesteuerten Simulation (PDES)* dar.

In den vergangenen Jahren wurde eine Vielzahl von Verfahren vorgestellt, die zur Lösung dieses Problems eine Synchronisation der (sequentiellen) Simulatoren für die einzelnen Teilmodelle gewährleisten. Neben einigen grundlegenden konzeptionellen Arbeiten wurden jedoch oft nur Teilaspekte der Problematik (z.B. Optimierung existierender Algorithmen) untersucht und die Frage, ob generell für diese Simulationsklasse Beschleunigungen in größerem Umfang erzielt werden können, bisher noch nicht geklärt.

An diesem Punkt setzt die vorliegende Arbeit an. Ihr Ziel ist es, die kritischen Einflußfaktoren für das Beschleunigungspotential der PDES zu isolieren, deren Auswirkungen auf die bekannten Simulationsverfahren zu ermitteln, Engpässe und Defizite zu erkennen und zu überprüfen, ob diese eliminiert oder reduziert werden können.

Da insbesondere auch die Relevanz der Ergebnisse für praktische Anwendungen und die Skalierbarkeit für große Modelle von Interesse ist, wurde als Szenarium für eine Evaluierungsumgebung existierender und weiterentwickelter Verfahren die *Simulation digitaler Schaltkreise* gewählt. Das prototypische Testbett DVSIM (**D**istributed **V**HDL **Sim**ulator) gestattet die Realisierung unterschiedlicher paralleler Simulationsverfahren. Zunächst werden die bekannten Basisprotokolle zur Synchronisation der Simulatoren (*konservatives* Verfahren nach Chandy/Misra/Bryant [CHM81a], [BRY79a] und sogenannte *Time Warp* von Jefferson [JES85a] als *optimistische* Methode) implementiert sowie deren Leistungsfähigkeit bewertet. Im Rahmen dieser Untersuchungen stellte sich heraus, daß die klassischen Basisalgorithmen keine guten Ergebnisse liefern; oft konnte damit zunächst überhaupt keine Beschleunigung erzielt werden.

Um realistische Aussagen bezüglich des Beschleunigungspotentials treffen zu können, wurde in der vorliegenden Arbeit ein *faires Bewertungsverfahren* eingeführt, in dem die Laufzeiten zwischen der parallelen Verfahren mit denen eines rein sequentiellen Simulators, der das gesamte Modell alleine bearbeitet, verglichen. Im Gegensatz dazu wird in vielen Veröffentlichungen von höheren Beschleunigungsraten berichtet, weil dort ungerechtfertigterweise Vergleiche relativ zur „verteilten“ Simulatorversion, die auf lediglich einem einzigen Prozessor lief, erfolgten. Wegen des selbst bei Verwendung nur eines Prozessors inhärenten Overheads einer verteilten Version fallen die Beschleunigungswerte dann fälschlicherweise auch höher aus.

Es zeigte sich weiterhin, daß die *Partitionierung des Modells*, also die Aufteilung der einzelnen Schaltkreiselemente auf die Teilsimulatoren, in Zusammenspiel mit den Synchronisationsverfahren eine größere Rolle als allgemein erwartet spielt. Wesentliche Beschleunigungen wurden allerdings erst bei großen Schaltkreisen registriert, für die sich der Parallelisierungsoverhead, der durch Kommunikation zwischen den sequentiellen Simulatoren und ihre notwendige Synchronisation entsteht, amortisiert. Ähnliches gilt auch für die Rechenzeit der Ereignisbearbeitungsroutinen. Wird die Granularität der einzelnen Ereignisbearbeitungsroutinen künstlich variiert, so zeigen sich für reale und synthetische Benchmarks bei höherer Granularität beachtliche Beschleunigungen. Bei Schaltkreisen aus realen Entwürfen konnte jedoch auch unter diesen modifizierten Bedingungen keine eindeutige Aussage zugunsten bestimmter Partitionierungsverfahren bezüglich einer im allgemeinen hohen Leistungsfähigkeit getroffen werden.

Da die Güte der Ergebnisse bei mehreren implementierten statischen Partitionierungsverfahren stark streute und die Partitionierungskosten bei großen Modellen i.a. nicht mehr tragbar erscheinen, wenn man qualitativ gute Aufteilungen erreichen will, wurden *dynamische Lastverteilungsverfahren* untersucht und hierfür das Simulationssystem DVSIM um eine entsprechende Komponente erweitert. Diese soll zum einen eine kostspielige initiale Partitionierung erübrigen und zum anderen und im Gegensatz zur statischen Partitionierung die Berücksichtigung des dynamischen Verhaltens der Simulation zur Laufzeit gestatten. Dazu werden Objekte während der Simulation von stark zu wenig belasteten Simulatoren verschoben. Als Lastmaß dient hier der Grad der Abweichung der lokalen Simulationszeiten von einem approximierten Minimalwert aller Uhren. Die Untersuchungen mit den so modifizierten Prototypen ergeben bei Verwendung des optimistischen Synchronisationsprinzips allerdings keine Verbesserungen. Der Aufwand für die dynamische Lastbalancierung führte in vielen Fällen sogar zu Leistungseinbrüchen. Für die konservativen Verfahren wurde ein mögliches Szenario zur dynamischen Lastbalancierung beschrieben, das aber aufgrund seiner zu erwartetenden Komplexität nicht umgesetzt wurde.

Abschließend betrachtet läßt sich feststellen, daß die Parallelisierung der ereignisgesteuerten Simulation offenbar nicht die in den 80er Jahren allgemein erwarteten Ergebnisse liefern kann. Es sind in der Praxis höchstens moderate Beschleunigungsfaktoren im Bereich zwischen 25 und 50 Prozent der optimalen (linearen) Beschleunigung möglich. Zudem kostet es relativ viel Wissen im

Bereich paralleler Programmiertechniken sowie einen hohen zeitlichen Aufwand, effiziente Simulatoren zu realisieren, die zumindestens diese Ergebnisse erzielen. Die Programmierung leistungsfähiger Simulatoren für einzelne Anwendungsklassen kann von einem Anwender ohne diesen vorhandenen Hintergrund nur schwer geleistet werden.

Allerdings hat sich auch gezeigt, daß ab einer gewissen Modellgröße und Ereignisgranularität Beschleunigungen nicht nur auf eng gekoppelten Parallelrechnern sondern auch auf Netzwerken aus Standardarbeitsplatzrechnern erreichbar sind. Dadurch bietet sich einem Anwender mit erzielbaren Beschleunigungsfaktoren von 2 - 5 (für sonst oft Tage oder gar Wochen benötigende Langläufer) bereits eine interessante Optimierungsperspektive.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung und Motivation

1.1 Modellbildung und Simulation

Menschen bemühen sich bereits sehr lange, sich die Welt, in der sie leben, mit dem ihnen zur Verfügung stehenden Wissen zu erschließen. So entstanden zu den verschiedensten Zeiten die unterschiedlichsten Erklärungsversuche für beobachtete Phänomene und Vorgänge in Natur und Technik. Die Erklärungsversuche basieren meistens auf einem Modell, das sich im allgemeinen aus (neuen) Beobachtungen und bereits vorhandenem Kontextwissen ableitete. Sie zielen darauf, Voraussagen für noch nicht beobachtete Fälle treffen zu können. Jedes Modell ist dabei im Kontext des zeitlichen Standes der Wissenschaft zu sehen.

Modelle basieren somit auf Axiomen, die als unumstößliche Voraussetzungen zunächst nicht hinterfragt werden. Aufgrund neuer Erkenntnisse und Beobachtungen können solche Axiome jedoch invalidiert, als unzutreffend erkannt oder als nicht ausreichend erachtet werden. So wurde beispielsweise im 15. Jahrhundert das geozentrische Weltbild durch ein heliozentrisches Modell abgelöst. Um die Erfaßbarkeit eines Modells zu ermöglichen, wird bei seiner Erstellung in der Regel von Details abstrahiert, die entweder nicht relevant für die Betrachtung und Beobachtung erscheinen, oder die nicht genau erfaßbar und beschreibbar sind. Voraussetzung für die Nutzbarkeit eines Modells ist aber stets, daß es in wesentlichen Aspekten das Verhalten des beobachteten Systems so exakt wie möglich (bzw. gewünscht) nachbildet. Die Entstehung eines Modells erfolgt dabei oft in mehreren Iterationen, in denen eine Überprüfung, Verfeinerung und Korrektur des Modells stattfindet [FIS95a].

Modellbildung läßt sich in zwei große Bereiche unterteilen. Einerseits besteht die Möglichkeit, ein reales System durch mathematische Sachverhalte und Formeln (z.B. in Abhängigkeit von Zeit und/oder Raum als freie Variablen) zu beschreiben. Durch Belegung der Formelvariablen mit entsprechenden Werten, die den auf das reale System wirkenden Reizen (Stimuli) entsprechen, wird eine bestimmte Vorhersage über das Verhalten (Auswirkungen an bestimmten Beobachtungspunkten) des realen Systems getroffen. Diese zunächst nur theoretischen Ergebnisse müssen zur Validierung des Modells mit beobachteten Daten abgeglichen werden.

Die zweite Möglichkeit zur Vorhersage des möglichen Systemverhaltens besteht in der Abbildung eines weitgehend unbekannten Systems (Zielsystems) auf ein anderes System (Originalsystem), dessen Verhalten bereits weitgehend verstanden wird (z.B. elektrische Schwingung und Dämpfung auf mechanische Schwingungs- und Dämpfungsmodelle). Diese Vorgehensweise kann weitestgehend mit dem Begriff *Simulation* bezeichnet werden. Dazu werden entweder die Formeln, die das Originalsystem beschreiben, für das Zielsystem angepaßt, oder es wird anhand eines Modell- und Experimentieraufbau des Experimentalsystems versucht, das Zielsystem zu verstehen. Durch Beobachtung des

bekannten Systems sind Rück- und Analogschlüsse auf das Verhalten des Zielsystems möglich.

Ein Zielsystem kann somit durch ein Originalsystem simuliert werden, indem von verschiedenen Startzuständen ausgehend die Zustände des Originalsystems beobachtet und seine Start- und Zielzustände mit denen des Zielsystems assoziiert werden. Die Beobachtung eines Systems besteht aus der Verfolgung einer Reihe von Zustandsübergängen (Transformationen). Ein Zustand wiederum besteht meist aus mehreren einzelnen Größen (Zustandsvariablen), die das System in seinen relevanten Aspekten vollständig beschreiben.

Das Wort Simulation bezeichnet dabei ein „Vorspiegeln von (falschen) Tatsachen“ (lat. simulatio) [DUD91a]. Man kann also bei einer guten Simulation den Eindruck bekommen, das nicht existente Zielsystem vor Augen zu haben, obwohl sein Verhalten durch ein anderes System nachgebildet wird. Die Güte der Approximation ist in diesem Zusammenhang ein oft subjektives Qualitätsmerkmal, das die Übereinstimmung von realem System und Modell bezeichnet.

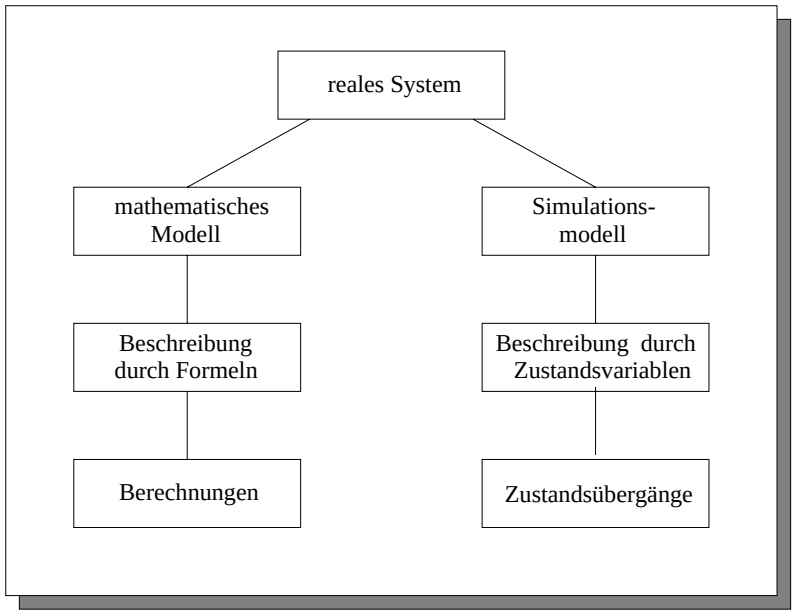


Abbildung 1.1: Modellierungsmöglichkeiten

Der erstgenannte Ansatz, reale Systeme mittels Formeln zu beschreiben, wird mit zunehmender Größe und Komplexität des Systems sehr schwierig. Geschlossene Ansätze, die eine effiziente analytische Betrachtung des Systemverhaltens gestatten, sind meist nur auf kleine Systeme anwendbar und liefern durch zunehmende Abstraktion und Vernachlässigung bestimmter Parameter (zur Verbesserung der Lösbarkeit) schnell eine unscharfe und nicht mehr vollständige Beschreibung des Systemverhaltens.

Simulation ermöglicht dem Modellierer dagegen, den Abstraktionsgrad und Betrachtungswinkel recht individuell nach seinen Wünschen festzulegen. Außerdem lassen sich sehr komplexe Verhalten, die analytisch nur schwer in Formeln zu fassen sind, durch Simulation gut beschreiben. Dazu wird das zu simulierende System schrittweise bis zum gewünschten Detailierungsgrad in elementare Objekte zerlegt, deren Verhalten mit der gewünschten Genauigkeit beschreibbar ist. Anhand der Eingangsbelegungen (Stimuli) dieser Objekte lassen sich nun iterativ Zustandsübergänge berechnen, die als Gesamtheit das Verhalten des realen Systems beschreiben.

Mit zunehmender Modellgröße steigt dann, je nach Detailierungsgrad, lediglich der Aufwand zur Ermittlung der Zustandsübergänge vom initialen Startpunkt bis zum Erreichen eines gewünschten

Endzustands an. Die exakte Nachbildung des Verhaltens bleibt jedoch weitgehend erhalten. Trotzdem stellt die Berechnung der Zustandsübergänge eine recht mühselige Angelegenheit dar, für deren Erledigung sich schnelle Computer eignen. Ähnliches gilt auch für die analytische Berechnung von Modellen, was aber das oben beschriebene Problem der Komplexität und Nichtverfügbarkeit geschlossener Lösungen für große Modelle nicht kompensiert.

Computergestützte Simulation hat heute in viele Bereiche Einzug gehalten. Dazu zählen Wettervorhersagen, Strömungssimulationen, Simulation von physikalischen und chemischen Vorgängen, Simulationen im militärischen und taktischen Bereich (hier wurden in der Vergangenheit viele grundlegende Arbeiten durchgeführt und viele Erkenntnisse gewonnen), Simulationen ökonomischer Vorgänge oder Simulation von Materialfluß- und Produktionssystemen.

Insbesondere im letztgenannten Gebiet dient die Simulation zur Validierung und Überprüfung von Modellen und geplanten Produkten auf Funktionsfähigkeit und Praktikabilität. Beispiele hierfür sind der computergestützte Rechnerentwurf und die Simulation von Computernetzen. Die Daten für die Simulationsmodelle stammen dabei oft aus computergestützten Entwurfsumgebungen, wodurch eine Simulation ihres Verhaltens auf einem Rechner naheliegt.

Demgegenüber stellt die Umsetzung in einen realen Prototyp oder ein äquivalentes Modell, sofern das überhaupt möglich ist, bei den sich drastisch verkürzenden Entwicklungszyklen mittlerweile keine Alternative mehr dar, um Produkte konkurrenzfähig am Markt anbieten zu können. Weiterhin entstehen auch hohe Kosten bei der Fertigung von Modellen. Daneben werden bei Simulationen durch Rechner ggf. Probleme auch frühzeitig erkannt noch bevor ein teurer Prototyp gefertigt wurde. Diese Vorgehensweise bedeutet wiederum Zeit- und Kostenreduktion und garantiert eine schnelle Integration der Erkenntnisse in weitere Iterationen des Entwicklungszyklus.

Ein weiterer Grund, der Simulation nötig macht, ist die Tatsache, daß reale Systeme oft sehr sensibel sind und ein Experiment ein solches System in einen unkontrollierbaren Zustand versetzen, es in irreparabler Weise schädigen oder zumindestens sehr hohe Kosten für die Kontrolle der Auswirkungen erzeugen könnte. Beispiele dafür sind kernphysikalische Vorgänge oder wirtschaftliche Zusammenhänge (Börse und Aktienmarkt). Eine explizite Steuerung solcher Systeme ist zudem oft gar nicht möglich. Ferner wird Simulation eingesetzt, um den zeitlichen Ablauf sehr schneller und sehr langsamer sowie mikro- und makroskopischer Vorgänge beobachtbar zu machen. Beispiele dafür sind chemische Reaktionen oder erdgeschichtliche Abläufe.

Eine ganz spezielle Klasse von Simulationen stellt schließlich die Nachbildung von Warteschlangenmodellen dar. Durch ihre mathematisch fundierte Theorie wird hier oft der Bogen zu analytischen Lösungsverfahren geschlagen. Dadurch können (zumindest für kleine Modelle) Simulationsergebnisse überprüft und auch vorausberechnet werden.

In den vergangenen Jahren sind im Zusammenhang mit der zunehmenden Bedeutung des Worldwide-web verstärkte Aktionen auf dem Gebiet der Simulation zu beobachten, die unter der Bezeichnung *Web-based-simulation* [FIS99a], [MIT99a] Applikationen mit Interfaces zur Steuerung über das WWW anbieten und so die Nutzung auch räumlich entfernter Ressourcen über das neue Medium gestatten.

Verstärkt werden diese Tendenzen durch das Aufkommen von Virtual-reality-Applikationen, die durch gekonnte graphische Animation von Handlungsabläufen die Illusion einer virtuellen Welt schaffen. Die graphische Komponente besitzt hier eine fundamentale Bedeutung, da die Qualität einer solchen Anwendung in erster Linie an den visuellen Eindrücken des Benutzers gemessen wird. Erfolgt die Betrachtung unter Realzeitbedingungen (z.B. Flugsimulator für Pilotentraining), so stellen sich sehr hohe Anforderungen an die benötigte Rechenleistung. Weniger zeitkritisch erscheint in diesem Bereich die Simulation in Entwurfssystemen für Architekten und Ingenieure, die das fertige Produkt animieren, eine weitgehend statische Struktur zugrundelegen und ein Navigieren

durch das Modell mit verschiedenen Blickwinkeln gestattet. Als Modellierungssprache hat hier in den vergangenen Jahren VRML (Virtual-reality-modeling-language) an Bedeutung gewonnen.

Des weiteren unterstützt insbesondere das amerikanische Verteidigungsministerium bereits seit Mitte der 80er Jahre Bestrebungen, eine einheitliche Schnittstelle zur Definition und Koppelung unterschiedlicher Simulationsobjekte und -modelle zu schaffen [DMS98a], um deren Interoperabilität und Wiederverwendbarkeit zu garantieren. Diese Bestrebungen kommen unter dem Namen *Distributed-interactive-simulation (DIS)* [SIS99a] insbesondere in Gefechtsfeldsimulationen zum Einsatz. Die beteiligten Simulatoren realisieren Kommunikation und Datenaustausch dabei über definierte Protokolle und Schnittstellen im Rahmen einer *High-level-architecture (HLA)* [DMS99a]. In [FUJ95a] werden die Anknüpfungspunkte dieser Bereiche mit PDES diskutiert.

Die Bedeutung der computergestützten Simulation kommt ebenfalls in der Existenz spezieller simulationsorientierter Programmiersprachen zum Ausdruck. Dazu zählen vor allem Simula [BDM73a], GPSS [SCH74a] und SIMSCRIPT II.5 [KMV83a]. Mit ihrer Hilfe können spezielle Modellkomponenten wie Prozesse und simulierte Objekte direkt in einer Programmiersprache beschrieben und verwaltet werden. Der Entwickler kann sich somit auf die Erstellung des Modells konzentrieren, ohne beispielsweise spezielle Ereignislistenverwaltung oder Schedulingkomponenten explizit programmieren zu müssen.

Bevor im folgenden auf Methoden für die computergestützte Simulation eingegangen wird, soll hier noch auf einen wichtigen Aspekt zum Begriff Simulation hingewiesen werden. Vielfach wird im allgemeinen Sprachgebrauch unter Simulation lediglich die graphische Darstellung von (Modell-) Verhalten im Sinne des genannten Virtual-reality-Ansatzes verstanden. Vom eigentlichen Verfahren, das die Ausgangsdaten für die graphische Darstellungen bereitstellt, wird dabei vollständig abstrahiert. Gerade auf die Methoden, nach denen die Modellzustände berechnet werden, wird aber in der vorliegenden Arbeit eingegangen. Die graphische Darstellung der Ergebnisse wird dabei eher den Bereichen Animation und Visualisierung zugerechnet, deren wirkungsvolle, aussagekräftige und gekonnte Realisierung insbesondere unter Realzeitbedingungen ein sehr komplexes Themengebiet darstellen.

1.2 Gründe für Beschleunigungsversuche

Durch die Verwendung von CAD-Entwurfsmethoden werden zunehmend größere Modellbeschreibungen handhabbar, die es zu simulieren gilt. Zusammen mit dem Wunsch detailierterer Modellierung und somit zusätzlich steigender Modellgröße verlangen die Anwender nach immer leistungsfähigeren Simulationsverfahren, die es gestatten, Ergebnisse schneller und präziser zu generieren, um sie in den Entwurfszyklus einbringen zu können. Im Bereich des Entwurfs höchstintegrierter Schaltkreise (VLSI) zeigt sich ein besonders starker Bedarf, der mit weiterer Verfeinerung der Entwurfsraster noch zunehmen wird. Um Simulationsergebnisse schneller bereitstellen zu können, werden dort bereits hierarchische Verfahren zur Simulation und Verifikation von Teilmodellen eingesetzt. Anschließend erfolgt lediglich die Simulation der Kooperation dieser Komponenten. Diese Vorgehensweise kann jedoch später beim realen Zusammenspiel der Teile zu teuren, nicht entdeckten Fehlern führen, weil von bestimmten Verhalten beim gröberen Simulationsmodell abstrahiert wurde. Der Wunsch nach detailgetreuer Simulation bleibt also trotz solcher Hilfsmittel und Verfahren bestehen.

Es ist somit nicht verwunderlich, daß im Simulationsbereich (insbesondere für Animationen) oft Höchstleistungsrechner mit hohen Kapazitäten eingesetzt werden. Ein Nachteil solcher Maschinen sind allerdings die hohen Anschaffungs- und Betriebskosten. Deshalb stehen sie oft nur einem eingeschränkten Nutzerkreis zur Verfügung. Mit dem Aufkommen von günstigeren (wenn auch nicht

billigen) Parallelrechnern Ende der 70er Jahre entstanden viele Forschungsprojekte, die sich enthusiastisch mit den Möglichkeiten der Beschleunigung von Simulationsläufen durch Parallelisierung und mit der Abbildung existierender Simulationsalgorithmen auf diese Architekturen befaßten. Allerdings setzten sich viele Parallelrechner aufgrund teilweise exotischer Architekturen und kleiner Stückzahlen nicht durch. Sie fristen eher ein Nischendasein im akademischen Bereich und finden wenig Anklang bei Firmen. Nichtsdestotrotz wurden in dieser Periode bis Anfang der 90er Jahre interessante Erkenntnisse auf dem Gebiet der parallelen Simulation gewonnen.

Das Aufkommen und die Vernetzung einfacher, aber leistungsfähiger Arbeitsplatzrechner sowie die Entwicklung von Standard-Kommunikationssoftware (z.B. MPI, PVM) zu deren Vernetzung als leistungsfähige virtuelle Parallelrechner bietet seit einiger Zeit jedoch einen neuen Ansatz zur Nutzung der in den 80er Jahren entwickelten parallelen Verfahren. Durch die Koppelung solcher Rechner über schnelle Kommunikationsnetze lassen sich auch räumlich verteilte Ressourcen miteinander verbinden. Diese Vorgehensweise wird im Gegensatz zur rein räumlichen Konzentration von Rechenleistung in Supercomputern als Meta-Computing bezeichnet. Selbstverständlich können hierbei auch Supercomputer als Rechnerressourcen mit eingebunden werden.

1.3 Zielsetzung dieser Arbeit

Ein Manko bisheriger Arbeiten auf dem Gebiet der parallelen Simulation ist jedoch die Nichtvergleichbarkeit vieler Ansätze. Es wurden Verfahren entwickelt, die auf verschiedensten Hardwareumgebungen implementiert, evaluiert und bewertet wurden. Eine Einordnung der Verfahren in ein (wenigstens einigermaßen geordnetes) Klassifikationsschema war bereits aufgrund der Verschiedenheit der Zielhardware und Architekturen nicht möglich. Heutige Implementierungen auf der Basis portabler Bibliotheken können diese Bewertung und gegenseitige Abgrenzung der Verfahren vielfach erleichtern.

Ein weiteres Problem stellt die Tatsache dar, daß existierende Arbeiten oft nur einzelne Teilaspekte betrachten. So erfolgen oft Evaluierungen der Einsetzbarkeit einzelner Synchronisationsverfahren, oder es existieren theoretische Abschätzungen der Kosten paralleler Simulationsverfahren, ohne daß jedoch dadurch die Frage beantwortet wird, ob sich parallele Simulation bezüglich erreichbarer Beschleunigung der Simulationsläufe überhaupt lohnt, beziehungsweise, unter welchen Voraussetzungen sich wenigstens ein rentables Verhältnis zwischen zusätzlichem Aufwand und erzieltem Nutzen einstellt.

Es kann natürlich nicht Sinn einer einzigen Arbeit sein, und es ist wohl auch nicht in einem solchen Rahmen durchführbar, den gesamten Bereich aller Varianten der parallelen Simulation komplett zu untersuchen. Dennoch wurde im vorliegenden Projekt eine sehr wichtige Klasse von Simulationsverfahren, die diskrete ereignisgesteuerte Simulation, in einem breiten Ansatz unter Einbeziehung bisheriger Forschungsergebnisse und neu gewonnener Erkenntnisse hinsichtlich ihrer Beschleunigungskapazitäten untersucht. Insbesondere sollen Einflußfaktoren identifiziert werden, die sich förderlich oder hinderlich darauf auswirken, und es soll eine Beurteilung existierender Bewertungsverfahren anhand einer protoypischen Implementation zur Simulation von VLSI-Schaltkreisen erfolgen. Aus den Ergebnissen dieser Betrachtungen werden schließlich Rückschlüsse auf die allgemeine praktische Einsetzbarkeit der parallelen diskreten ereignisgesteuerten Simulation gezogen.

Im folgenden Kapitel werden die Prinzipien der diskreten ereignisgesteuerten Simulation gegen andere Simulationsverfahren abgegrenzt und eine Beschreibung und Klassifizierung existierender Verfahren vorgenommen. Kapitel 3 definiert Methoden, mit denen die Leistungsfähigkeit paralleler Simulationsverfahren in der vorliegenden Arbeit beurteilt werden. Die Spezifika, die bei der verteilten Schaltkreissimulation beachtet werden müssen, folgen in Kapitel 4. Kapitel 5 diskutiert

Vorgehensweisen zur Aufteilung eines Simulationsmodells in mehrere (disjunkte) Teilmodelle, die der parallelen Simulation vorausgehen muß, und schließt Teil I dieser Arbeit ab.

Teil II beschreibt die durchgeführten Untersuchungen zur parallelen Simulation anhand der prototypischen Implementierung des Schaltkreissimulators DVSIM, dessen Aufbau in Kapitel 6 dargelegt wird. Kapitel 7 befaßt sich mit den Fragestellungen und angestrebten Zielen, die mit Hilfe der Untersuchungen am Prototyp angegangen wurden, und den Methoden, die bei den Untersuchungen zum Einsatz kamen. Die Ergebnisse, die mit den parallelen Implementierungen erreicht wurden, sind Thema von Kapitel 8. In Kapitel 9 wird ein Verfahren zur dynamischen Lastbalancierung zur Laufzeit des Simulators als Versuch einer weiteren Optimierung vorgestellt und seine Leistung bewertet.

In Kapitel 10 erfolgt eine Beurteilung der Skalierbarkeit der Ergebnisse bezogen auf die Modellgröße und verschiedenen Rechnerplattformen (Parallelrechner und Netz von Arbeitsplatzrechnern). Die Ergebnisse dieser Arbeit werden in Kapitel 11 diskutiert. Ausblicke auf offene gebliebene Fragen und eventuell weiter zu untersuchende Aspekte beschließen die Arbeit.

Teil I

Parallele Simulation: Ein offenes Feld

Kapitel 2

Simulationsmethoden

Dieses Kapitel beginnt mit einer Klassifizierung verschiedener Simulationsmethoden. Danach werden Möglichkeiten zur Beschleunigung der rein sequentiellen Simulation beschrieben, wobei insbesondere auf die ereignisgesteuerten Varianten eingegangen wird. Wichtige, bei der Parallelisierung auftretende Probleme werden dargestellt und einige dafür vorgeschlagene Lösungen diskutiert.

2.1 Klassifikation

Simulationsverfahren, die die Zeit als freie Variable verwenden, weisen zwei wesentliche Kriterien zu ihrer Charakterisierung auf. Als erstes Merkmal dient die Unterscheidung des zeitlichen Verlaufs der Werte von Zustandsgrößen. Das zweite Kriterium bestimmt die Art der Zeitfortschaltung.

2.1.1 Zeitmodellierung

Die Zeit läßt sich als kontinuierliche oder diskrete Größe modellieren. Im ersten Fall treten Zustandsänderungen des Simulationsmodells als mehr oder weniger gleichmäßige Prozesse über die gesamte Simulationsdauer hinweg auf. Die Zustandsgrößen verändern sich kontinuierlich mit der Zeit. Zu jedem Zeitpunkt läßt sich ein exakter Werte für alle relevanten Größen angeben, der dem Wert im zugrundeliegenden realen System entspricht.

Die diskrete Modellierung der Zeit betrachtet demgegenüber das Simulationsmodell nur zu bestimmten diskreten Zeitpunkten, die für das Simulationsergebnis relevant sind. Meist wird dazu ein Zeitraster eingeführt, dessen Auflösung Δt die minimalen Abstände zwischen relevanten Zustandsübergängen respektieren muß. Innerhalb der Zeitintervalle zwischen zwei benachbarten Zeitpunkten sind keine Zustandsänderungen vorgesehen. Die Zustandsvariablen weisen für diese (halboffenen) Intervalle einen konstanten Wert auf, der von den Werten am linken Intervallrand bestimmt wird. Die Werte der Zustandsgrößen repräsentieren zu diesem Zeitpunkt bereits die vollzogenen Änderungen.

Betrachtet man die Werte einer Zustandsgröße X als Graph in Abhängigkeit von der Zeit, so ergibt sich bei kontinuierlicher Modellierung typischerweise eine stetige Funktion, während eine diskrete Zustandsvariable viele Sprungstellen aufweist und zwischen zwei Zeitpunkten einen konstanten Wert behält (Abb. 2.1).

Kontinuierliche Verfahren werden oft zur Simulation physikalischer Vorgänge mit kontinuierlichen Größen verwendet. Dazu zählen z.B. Strömungsverhalten in Flüssigkeiten und Gasen oder Betrachtungen analoger Strom- und Spannungspegel elektrischer Schaltungen. Als Modellierungsverfahren eignen sich insbesondere Differentialgleichungssysteme.

Diskrete Verfahren können zunächst (hinsichtlich der algorithmischen Behandlung) als Vereinfachung der kontinuierlichen betrachtet werden. Die kontinuierliche Funktion wird quasi mit dem vorgegebenen Zeitraster Δt abgetastet und die ermittelten Werte bis zum nächsten Abtastzeitpunkt als konstant angenommen.

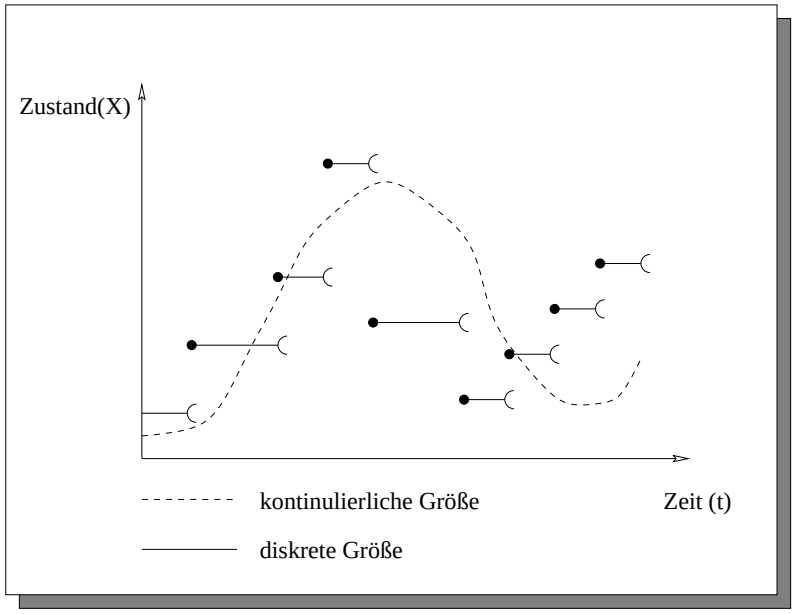


Abbildung 2.1: Zeitliches Verhalten von Zustandsgrößen

Die konkrete Realisierung folgt jedoch nicht diesem Schema sondern verwendet statt der Abbildung der kontinuierlichen auf eine diskrete Funktion eine eigene Zustandsübergangsfunktion, die aus dem Zustand zum Zeitpunkt t den Folgezustand zum Zeitpunkt $t + \Delta t$ ermittelt. Zweck dieser Vorgehensweise ist nicht eine lückenlose Aufzeichnung der Historie der Zustandsgrößen sondern die Erfassung der relevanten Zustände und Zustandsübergänge. Durch Verfeinerung des Zeitrasters läßt sich ein kontinuierliches Verfahren durch ein diskretes approximieren (quasi-kontinuierliches Verhalten).

2.1.2 Zeitfortschaltung

Das andere Unterscheidungsmerkmal für Simulationsverfahren ist die Methode zur Fortschaltung der Simulationszeit. Diese kann einmal entsprechend dem erwähnten festen Zeitraster Δt erfolgen, so daß die Zustandsgrößen stets für jeden Folgezeitpunkt aus dem vorhergehenden neu berechnet werden. Die Berechnung der Zustände ist dabei völlig unabhängig davon, ob sich im Modell tatsächlich Änderungen vollzogen haben, und wird *zeitgesteuert* durchgeführt.

Ein optimiertes Verfahren setzt eine etwas "intelligentere" Zustandsübergangsfunktion voraus, die den nächsten Zeitpunkt ermittelt, zu dem sich der Zustand der zugehörigen Größe ändern wird. Die Zeit wird also mitunter in variablen Vielfachen des Zeitrasters Δt stets zum nächsten Änderungszeitpunkt fortgeschaltet, so daß Zustände des Modells nur zu Zeitpunkten neu berechnet werden, an denen sich mindestens eine Größe ändert. Die zwischenliegenden Zeiträume ohne Aktivitäten (Totzeiten) werden übersprungen. Da dieses Verfahren durch das Eintreten bestimmter Ereignisse kontrolliert wird, verdankt es seiner Vorgehensweise die Bezeichnung *ereignisgesteuert*.

Diese Differenzierung nach der Art der Zeitfortschaltung läßt sich im wesentlichen nur auf dis-

krete Modellierungsverfahren anwenden. Kontinuierliche Verfahren unterliegen dagegen prinzipiell einer Zeitsteuerung mit idealerweise infinitesimal kleinem Wert von Δt.

Als weitere diskrete ereignisbasierte Verfahren von teilweise untergeordneter Bedeutung existieren die aktivitätsorientierten, prozeßorientierten und transaktionsorientierten Simulationsmethoden, die in [MaM89a] diskutiert werden, und auf die hier nicht näher eingegangen wird, da sie sich im wesentlichen auf die ereignisgesteuerte Simulation abbilden lassen und lediglich unterschiedliche Sichtweisen auf das simulierte System anbieten.

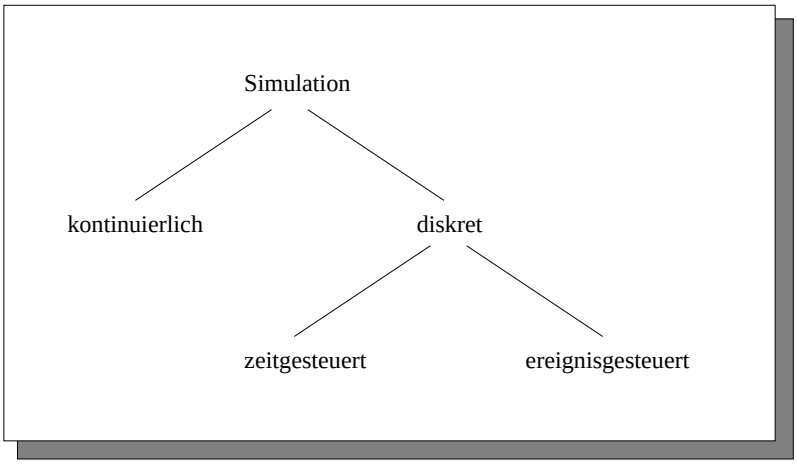


Abbildung 2.2: Zeitmodellierung und -fortschaltung bei Simulationsverfahren

Für die folgenden Betrachtungen spielt die diskrete ereignisgesteuerte Simulation die wesentliche Rolle. Abb. 2.2 stellt die vorgenommene Einteilung dar. Neben der hier vorgestellten Klassifikation geben Peacock, Wong und Maning in [PWM79a] eine weitere sehr detailierte Einordnung paralleler Simulationsverfahren an.

2.2 Die Methode der diskreten ereignisgesteuerten Simulation

2.2.1 Grundsätzliche Arbeitsweise

Zur Erstellung diskreter ereignisgesteuerter Simulationsmodelle muß das zu simulierende System auf die relevanten Objekte und ihre Interaktionsschemata abgebildet werden. Objekte werden dabei durch eine oder mehrere Zustandsvariablen repräsentiert. Je nach Abstraktionsgrad werden dabei nur diejenigen Eigenschaften, die für die Simulation von Bedeutung sind, erfaßt. Die Zustandsvariablen stellen jedoch lediglich den statischen Teil des Simulationsmodells dar. Das dynamische Verhalten der Simulation wird durch atomar eintretende Aktionen (Ereignisse) realisiert, die die Interaktion der Objekte widerspiegeln.

In der diskreten ereignisgesteuerten Simulation werden diese Interaktionen, die den Systemzustand eines oder mehrerer Objekte verändern können, als Ereignis betrachtet. Zugunsten eines übersichtlichen Designs empfiehlt es sich jedoch, Ereignisse lediglich einem einzigen Objekt zuzuordnen und komplexere Interaktionen durch mehrere objektbezogene Einzelereignisse darzustellen.

Ereignissen ist ein diskreter Eintrittszeitpunkt aber keine Dauer zugeordnet. Folglich schreitet die Simulationszeit durch die Ausführung eines Ereignisses nicht fort. Die Ausführung eines Ereignisses besteht in der Abarbeitung einer zugehörigen Ereignisroutine, die die Zustandsvariablen des Modells modifizieren und wiederum weitere Ereignisse zur Ausführung in der Zukunft generie-

ren kann (aber nicht muß). Mehrere zukünftige Ereignisse können sozusagen auf Halde produziert werden (Abb. 2.3).

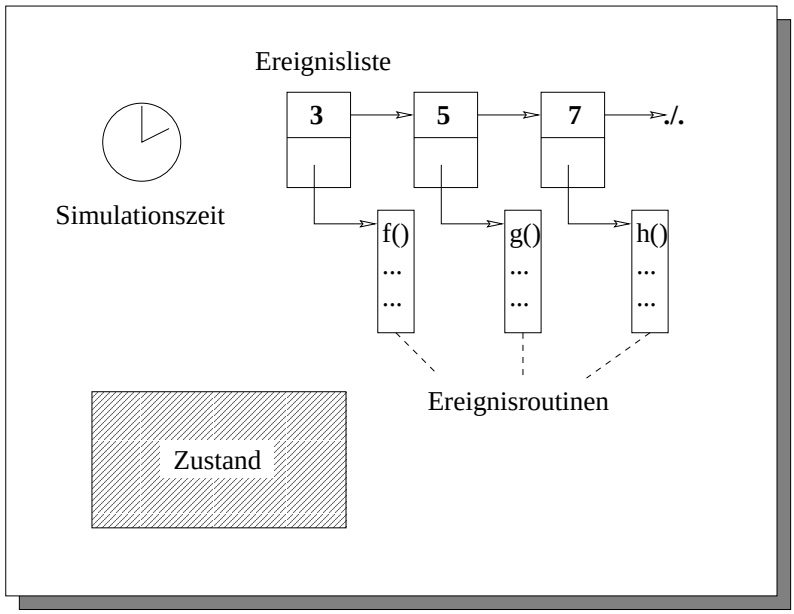


Abbildung 2.3: Simulationsmodell der diskreten ereignisgesteuerten Simulation

```
t sim = 0;
e = EvListe.entferne_kleinstes_Ereignis();
while ( e != null) loop
    t sim = e.Zeitstempel;
    neue_Ereignisse[ ] = e.Ereignisroutine();
    forall (ev in neue_Ereignisse[ ]) do
        EvListe.einfügen(ev);
    done
    e = EvListe.entferne_kleinstes_Ereignis();
endloop
```

Abbildung 2.4: Abarbeitungszyklus

Die Verwaltung der noch nicht ausgeführten Ereignisse erfolgt über eine nach Zeitstempel aufsteigend sortierte Verwaltungsdatenstruktur, die im folgenden unabhängig von ihrer konkreten Realisierung als *Ereignisliste* bezeichnet wird. In einem Zyklus wird stets das kleinste Ereignis aus der Liste entfernt, die Simulationsuhr auf den Zeitstempel dieses Ereignisses gesetzt und die entsprechende Ereignisroutine aufgerufen (Abb. 2.4).

Die Simulation endet, wenn die Ereignisliste leer wird oder ein vorgegebenes Limit wie z.B. eine bestimmte Simulationszeit überschritten ist. Diese Vorgehensweise impliziert, daß zu Simulationsbeginn wenigstens ein Ereignis initial vorhanden sein muß.

2.2.2 Beschleunigungspotentiale

Aus der hier vorgestellten Beschreibung der Realisierung (rechnergestützter) Simulation nach dem ereignisgesteuerten Prinzip ergibt sich im Hinblick auf die zu erwartende Berechnungsdauer des Simulationsmodells eine duale Betrachtungsweise des Zeitbegriffs.

Es gilt zwischen der (realen) Rechenzeit zur Ausführung des Simulationsmodells und der Simulationszeit zur Modellierung des Simulationsfortschritts im Modell zu unterscheiden. Das Verstreichen von Simulationszeit benötigt im Verhältnis zur Gesamtdauer der Berechnung (nahezu) keine Rechenzeit, da die Zeiträume zwischen zwei Ereignissen schlichtweg übersprungen werden.

Im Gegensatz dazu erfordern die Ereignisse, die für das Simulationsmodell atomar, d.h. in Nullzeit, geschehen, im Simulator die Ausführung der zugehörigen Ereignisroutinen, die die Zustandsübergänge und ggf. neue Ereignisse berechnen. Diese Berechnungen können mitunter sehr komplexe und rechenzeitaufwendige Funktionen sein. Vereinfacht ausgedrückt bedeutet dies:

> Was im Simulationsmodell Zeit benötigt, braucht bei der Rechnersimulation keine Zeit,
>
> und umgekehrt gilt auch:
>
> Was in der Rechnersimulation Zeit benötigt, erfordert im Simulationsmodell keine Zeit.

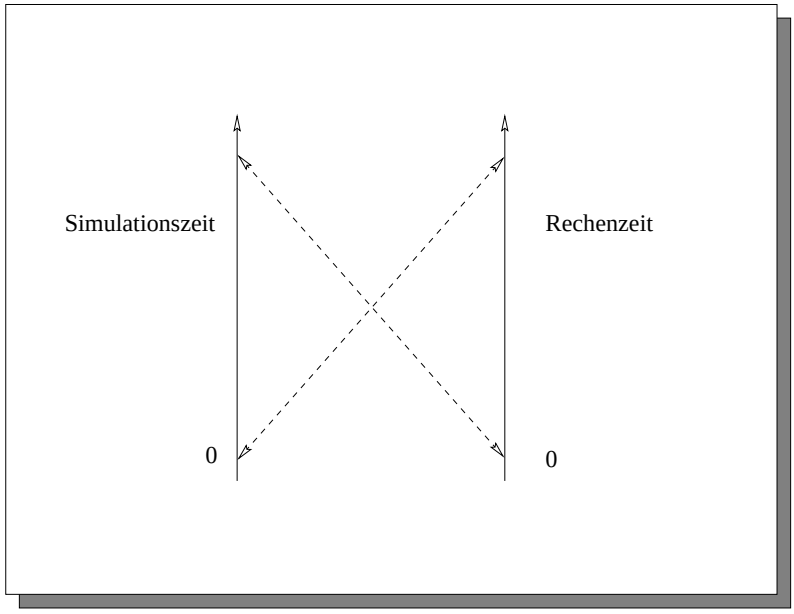


Abbildung 2.5: Dualismus der ereignisgesteuerten Simulation

Dieses Prinzip wird von Mattern und Mehl auch als *Dualismus der ereignisgesteuerten Simulation* [MaM89a] bezeichnet und ist in Abb. 2.5 nochmals verdeutlicht.

Somit ergibt sich die Gesamtlaufzeit einer ereignisgesteuerten Simulation als Summe der einzelnen Rechenzeiten für die Ausführung aller Ereignisroutinen bei sequentieller Abarbeitung der Ereignisse in aufsteigender Zeitstempelreihenfolge.

Die bisherigen Überlegungen zeigen, daß die Simulationsdauer etwa linear mit der Anzahl der zu simulierenden Ereignisse wächst. Da jedoch die Anzahl der Ereignisse mit zunehmender Detaillierung des Modells z.B. zur exakteren Berechnung der Ergebnisse dramatisch ansteigen kann, läßt sich die Komplexität vieler Modelle auch mit schnellen konventionellen Rechnern nur noch schwer beherrschen. Um den wachsenden Ansprüchen durch zunehmende Modellgröße, Detailierung und Simulationslänge Herr zu werden, existieren verschiedene Ansätze, die nun beschrieben werden.

2.3 Verfahren zur Beschleunigung von Simulationen

Die Begriffe *parallel* und *verteilt* werden insbesondere im Bereich der ereignisgesteuerten Simulation häufig ohne Unterscheidung und in der Literatur nahezu beliebig austauschbar benutzt. Dennoch sollte beachtet werden, daß Verteiltheit im Grunde eine räumliche Trennung der Komponenten, die zur Gesamtlösung beitragen, ggf. auch über größere Distanzen impliziert. Parallelität bezieht sich hingegen zwar auf eine Trennung in gleichzeitig ablaufende Komponenten, schränkt aber die (räumliche) Entfernung, in der solcher Anwendungsteile ablaufen, meist stark ein. Parallele Verfahren beschreiben deshalb eher ein eng gekoppeltes kooperatives Nebeneinander.

Eine naheliegende Idee, die Ausführung zu beschleunigen, besteht in der Übertragung der Berechnung von einem Prozessor, der die gesamte Simulation allein durchführt, zur arbeitsteiligen Verwendung mehrerer Prozessoren, von denen jeder nur eine Teilaufgabe bei der Gesamtsimulation lösen muß. Davon erwartet man sich eine insgesamt kürzere Bearbeitungsdauer. Voraussetzung ist dabei jedoch die Verfügbarkeit leistungsfähiger Mehrprozessorarchitekturen.

Vor der Beschreibung modifizierter und zur parallelen Berechnung angepaßter Simulationsmethoden sollte jedoch der Begriff Mehrprozessorrechner genauer definiert werden.

2.3.1 Klassifikation von Parallelrechnern

Parallelrechner besitzen einen engen Koppelungsgrad und verfügen meist über ein eigenes schnelles Kommunikationsnetzwerk oder über gemeinsamen Speicher zum Datenaustausch. Bei verteilten Systemen erfolgt im Gegensatz dazu die Koppelung der einzelnen Komponenten in der Regel über konventionelle Netze wie LANs oder WANs. Die unterschiedlichen Koppelungsmechanismen haben dadurch auch direkte Auswirkung auf die Geschwindigkeit, mit der die einzelnen Prozessoren Daten austauschen können.

Daß die Begriffe parallel und verteilt im algorithmischen Bereich dennoch oft synonym verwendet werden, liegt daran, daß sich viele konkrete Verfahren prinzipiell in beiden Architekturmodellen einsetzen lassen, wobei die Geschwindigkeit der Abarbeitung stark variieren kann.

Durch die Entwicklung virtuell paralleler Programmierplattformen wie p4 (protable programs on parallel processors) [BUL92a], PVM (Parallel-virtual-machine) [SUN90a, SGD93a] und MPI (Message-passing-interface) [MPI93a] wird diese ohnehin schon unklare Trennungslinie weiter verwischt. Im Bereich der Simulation wurde die Mehrzahl der Arbeiten bis vor wenigen Jahren auf echten Parallelrechnern realisiert. Trotzdem wurde vielfach (insbesondere in der englischsprachigen Literatur) der Term *distributed simulation* verwendet. Allerdings beansprucht unabhängig davon auch die Distributed-interactive-simulation (DIS) [FUJ95a] diese Bezeichnung für sich, so daß sich

mittlerweile doch der Begriff *parallel simulation* für die im folgenden beschriebenen Verfahren eingebürgert hat [FUJ90a, NIF94a].

Die Architekturen von Rechnern mit mehreren Prozessoren lassen sich anhand verschiedener Modelle unterscheiden:

- Flynn präsentierte ein Klassifikationsschema [HWB84a], nach dem sich das Verhalten an der Art und Weise der Programmsteuerung (gemeinsame, zentral gesteuerte oder autonome Ablaufkontrolle der Prozessoren) und der Anzahl der Datenströme, auf denen diese Kontrolle abläuft, orientiert. SIMD[1] und MIMD[2] sind die verbreiteten Vertreter von Rechnern nach Flynns Einstufung. Zur ersten Kategorie gehört bspw. die CM-2 mit bis zu 65536 Prozessoren [CM2]. Die zahlenmäßig weitaus größere zweite Gruppe beinhaltet u.a. Intel Hypercube iPSC/860 [INT91a], IBM SP-2 [SP298a], KSR-1 [KSR94a], Cray T3D und Sequent Symmetry [SEQ99a].

- Als weiteres Unterscheidungskriterium für Parallelrechner dient die Art der Organistation des Speicherzugriffs. Systeme, in denen alle Prozessoren auf gemeinsam genutzten Hauptspeicher zugreifen können, bezeichnet man als Shared-memory-Systeme (z.B. KSR-1). Je nachdem, ob für den gemeinsamen Speicher unterschiedliche Zugriffsverfahren bestehen, wenn der physikalische Ort der Speicherung berücksichtigt werden muß, oder ob der Zugriff völlig transparent erfolgt, heißen diese Architekturen NUMA bzw. UMA, (Non) Uniform-memory-access. Allerdings sind Shared-memory-Verfahren bekannt für ihre eingeschränkte Skalierbarkeit bezüglich der Prozessorenanzahl [SIS94a]. Existiert dagegen für jeden Prozessor ein nur für ihn lokal zugreifbarer Speicher, so spricht man von Distributed-memory-Systemen.

- Eng mit dem vorherigen Punkt spielt die Frage zusammen, wie Koordination, Kooperation und Synchronisation erfolgt. Bei Shared-memory kann der Datenaustausch selbstverständlich über gemeinsame Speicherbereiche nach gewissen Konventionen (Zugriffskontrolle, Sperrverfahren, wechselseitiger Auschluß) erfolgen. In Systemen mit verteiltem Speicher greift man auf Daten anderer Prozessoren durch Anforderung der Daten mittels Nachrichten zu. Das dazu benötigte Protokoll muß dabei von allen Beteiligten verstanden und eingehalten werden.

  Mit Hilfe von Nachrichten läßt sich auch auf einem System mit verteiltem Speicher die Illusion eines virtuellen oder verteilten gemeinsamen Speichers schaffen. Die Eleganz des transparenten Speicherzugriffs muß allerdings mit Effizienzverlusten aufgrund der das Verfahren realisierenden Nachrichten erkauft werden.

- Für die Leistungsfähigkeit der Parallelrechner spielt letztendlich auch das zugrundeliegende Kommunikationsnetz eine wesentliche Rolle. Als Topologien werden häufig Gitter, Hypercubes sowie Crossbar-switches und Permutationsnetzwerke verwendet. Besondere Bedeutung wurde dabei Mitte der achtziger Jahre dem Transputer zugemessen, der ebenso wie eine Reihe weiterer digitaler Signalprozessoren (z.B. C40 von Texas Instruments) über zusätzliche hardwaremäßig realisierte Kommunikationsverbindungen (Links) verfügt, über die mehrere dieser Prozessoren zu unterschiedlichen Topologien verschaltet werden können. Leider ist diese Technologie als Hauptprozessor eines Rechnerknotens mittlerweile überholt und kommt heute nur noch als Kommunikationskoprozessor in einigen Parallelrechnern vor [PPC96a].

[1] Single-instruction-multiple-data, auch SPMD (P = program)
[2] Multiple-instruction-multiple-data, auch MPMD

2.3.2 Parallele Simulationsmethoden

Nahezu ebenso vielfältig wie die Realisierungen von Parallelrechnern sind die Verfahren, nach denen Simulationsalgorithmen parallelisiert werden können.

2.3.2.1 Kontinuierliche Simulation

Im Bereich der kontinuierlichen Simulation haben parallele Gleichungslöser die fast uneingeschränkte Herrschaft. Es existieren für eine Vielzahl von Problemen bereits umfangreiche Bibliothekensammlungen, die zu einer Applikation lediglich hinzugebunden werden müssen. In Abhängigkeit von der Problemgröße nehmen sie die automatische Verteilung der Berechnungen auf Parallelrechner vor und garantieren mit effizienten Lösungen eine beschleunigte Abarbeitung der Aufgaben (z.B. ScaLAPACK [BCC96a]).

2.3.2.2 Zeitgesteuerte Simulation

Die diskrete zeitgesteuerte Simulation verwendet im wesentlichen ein zyklisches Zwei-Phasen-Modell zur Parallelisierung ihrer Algorithmen mit einer global getakteten Simulationszeit. Die Simulationsobjekte werden zunächst auf die einzelnen Prozessoren verteilt. Nach einem Startschuß berechnen alle Rechnerknoten die Zustandsveränderungen der lokalen Objekte im ersten Zeitabschnitt. Sind alle Berechnungen durchgeführt, können in einer zweiten Phase die Daten zwischen den Knoten ausgetauscht werden, die Zeit wird um das feste Inkrement Δt fortgeschaltet und die Simulation startet wieder mit der nächsten Iteration von Phase 1 (Abb. 2.6). Die Synchronisation der Berechnungs- und Kommunikationsabschnitte kann zentral oder dezentral gesteuert werden.

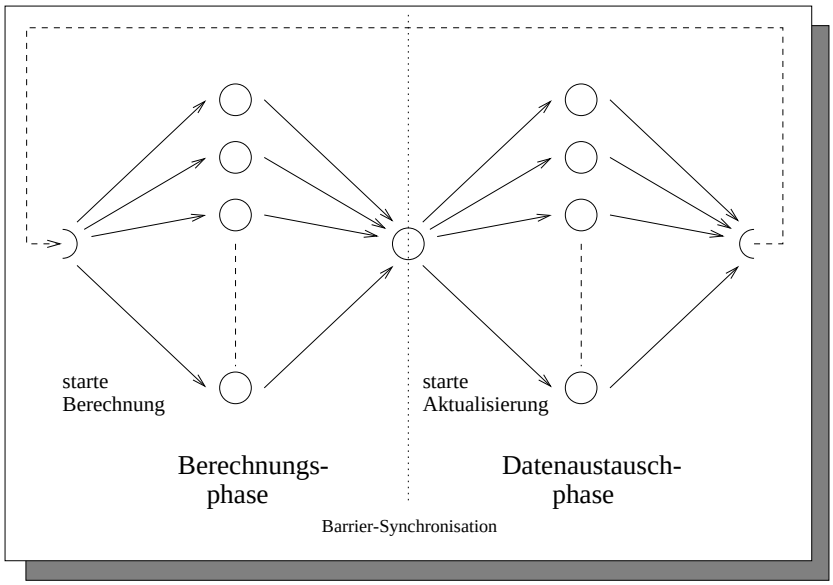


Abbildung 2.6: Bearbeitungsphasen der zeitgesteuerten Simulation

Auf der zeitgesteuerten Parallelisierungsvariante setzt eine Optimierung auf, die stellenweise in der Literatur ebenfalls als (synchron) ereignisgesteuert bezeichnet wird [SOU92a]. Dabei werden Zeiträume, in denen keine Aktivitäten im Simulationsmodell vorhanden sind, ebenfalls als Totzeiten übersprungen (Abb. 2.7). Die einzelnen Simulatoren müssen sich dazu während der Berechnungsphase den frühesten Zeitpunkt einer nachfolgenden Aktivität im System merken. Während der

Datenaustauschphase einigen sich die Simulatoren dann, auf welchen Zeitpunkt die immer noch global synchronisierte Uhr vorgestellt wird (Minimumbildung der lokalen Minima). Obwohl hierbei ein großer Teil nicht notwendiger Arbeit bereits eingespart werden kann, wird durch die starre Koppelung an Zeitraster durch die synchrone Uhr Beschleunigungspotential verschenkt. Nach diesem Prinzip arbeiten Lubachewskys *Bounded-lag-* [LUB89a] und Steinmans *Time-bucket*-Verfahren [STE91a].

Zur Veranschaulichung der Zusammenhänge werden im folgenden Zeitdiagramme verwendet, bei denen (wo nicht explizit anders erwähnt), die waagrechten Pfeile einzelne Prozessoren mit von links nach rechts fortschreitender Simulationszeit darstellen. Punkte auf diesen Achsen repräsentieren atomare Ereignisausführungen und Pfeile zwischen den Achsen stehen für Nachrichten, die sich die Prozessoren untereinander schicken. Das Ereignis am Fuß des Pfeils ist das Sendeereignis (oder generierendes Ereignis). An der Spitze eines Pfeils findet sich das generierte oder Empfangsereignis.

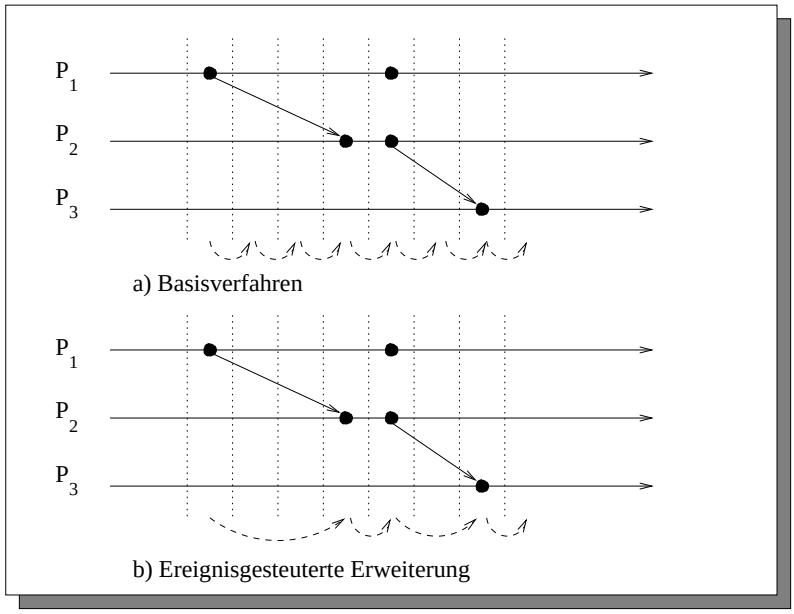


Abbildung 2.7: Alternativen bei der Zeitfortschaltung zeitgesteuerter Simulation

2.3.2.3 Ereignisgesteuerte Simulation

Die Parallelisierung der ereignisgesteuerten Simulation erlaubt hier gegenüber der zeitgesteuerten Varianten weitere Verbesserungen. Wie bereits in der Einführung dargelegt, benötigt die Berechnung der Zustandsübergänge in Ereignisroutinen die eigentliche Rechenzeit. Sind nun die Ereignisse, die abgearbeitet werden sollen, kausal unabhängig, das heißt, sie haben gegenseitig keinen direkten oder indirekten Einfluß aufeinander, weil sie nur disjunkte Zustandsvariablen benutzen, so spricht nichts dagegen, diese Ereignisse, auch wenn sie unterschiedliche Zeitstempel tragen, zur selben Zeit von zwei Prozessoren parallel bearbeiten zu lassen. Die Synchronität weicht in diesem Ansatz einem asynchronen Verfahren, in dem die globale Uhr durch lokale Uhren ersetzt wird. Diese können unabhängig voneinander mit beliebiger Geschwindigkeit (bestimmt von der Länge der Ereignisroutinen und übersprungenen Totzeiten) voranschreiten, sofern die Ereignisse auch über indirekte Wege keine globalen Kausalitätsverletzungen begehen. Es dürfen somit keine Zustandsvariablen geändert werden, die ein "früheres" Ereignis noch lesen muß.

Bei der parallelen diskreten ereignisgesteuerten Simulation wird das Gesamtmodell in Teilmodelle mit paarweise disjunktem Zustandsraum zerlegt. Jedes Teilmodell wird einem Prozessorknoten zugewiesen, auf dem quasi ein sequentieller Simulator läuft, der ausschließlich die Berechnungen für dieses Teilmodell ausführt (Abb. 2.8). Insoweit ist diese Vorgehensweise bei der Modellaufteilung analog zur zeitgesteuerten Variante.

Chandy und Misra prägten in ihren frühen Arbeiten [CHM81a] die grundlegenden Begriffe *physischer Prozeß (PP)* und *logischer Prozeß (LP)*. Ein physischer Prozeß entspricht den bereits bei der Modellierung erwähnte elementaren Objekten, die die relevanten Größen des betrachteten Systems beschreiben. Die logischen Prozesse bilden das in Software gefaßte und zur Simulation verwendete Äquivalent. Die sequentielle Abarbeitung innerhalb der logischen Prozesse bei gleichzeitig paralleler Aktivität mehrerer (aller) LPs und ihre asynchrone Kooperation über Nachrichten oder gemeinsame Speicherbereiche beschreiben in recht anschaulicher Weise das Grundprinzip der parallelen diskreten ereignisgesteuerten Simulation. Im folgenden wird, wo nicht anders erwähnt, stets die Kommunikation mittels Nachrichten zur Zusammenarbeit von LPs vorausgesetzt. Es lassen sich sogar Ereignisse, die ein LP lokal generiert und in die eigene Ereignisliste einplant, als Nachrichten des LPs an sich selbst betrachten. Dadurch erhält man ein vollständig homogenes Bild für die Generierung und Verarbeitung von Ereignissen.

Bei einer limitierten Anzahl von Prozessoren können prinzipiell auch mehrere Teilmodelle mit jeweils einem eigenen Simulatorprozeß den einzelnen Rechnerknoten zugewiesen werden. In den frühen Ansätzen wird jeder LP von einem eigenen Simulator ausgeführt. Bei den dort betrachteten geringen Anzahl von Objekten (z.B. bei Warteschlangenmodellen) [FUJ88a, FUJ88b] macht diese Vorgehensweise durchaus Sinn, da jedes Teilmodell prinzipiell einen eigenen Prozessor zugewiesen bekommt. Bei größeren Simulationsmodellen finden sich allerdings mitunter sehr viele LPs auf einem Rechnerknoten und teilen sich dessen Ressourcen.

Dadurch entsteht sehr hoher Aufwand für das Scheduling der einzelnen Simulatorprozesse. Zur Optimierung wurden verschiedene Verfahren unter Verwendung von Threads und eigener Scheduler unter Beibehaltung eines eigenen LPs für jedes einzelne elementare Objekte entwickelt. Der Scheduler plant dabei z.B. denjenigen Thread mit dem kleinsten ausführbaren Ereignis (Shortest-timestamp-first, STF) ein. Diese Strategie wurde z.B in der Simulationsumgebung YAWNS [NMI90a] realisiert, findet sich aber auch in anderen Ansätzen [AVT96a]. [LIL90b] entwickelt eine erweiterte preemptive Schedulingstrategie, die bessere Beschleunigungswerte im Vergleich zum reinen STF-Verfahren liefert.

Als Alternative bietet sich die Zusammenfassung aller auf einem Prozessor laufenden Simulatoren zu einem Cluster an, das ohne interne Prozeßverwaltung auskommt und die Datenstrukturen aller Objekte zusammenfaßt [RIC95a, LUK93a, SOU92a, SUS89a]. Man gibt also die Unabhängigkeit der einzelnen Simulatoren zugunsten der einfacheren Prozeßverwaltung auf. Da aber ohnehin nur ein einziger Prozeß pro Prozessor zu einem gegebenen Zeitpunkt aktiv sein kann, ist dieser Nachteil bei entsprechender Auslastung des Clusters nicht so gravierend. Andererseits kann aufgrund geringer Aktivität einiger weniger Objekte jedoch der gesamte Cluster blockieren, während beim Verfahren mit vielen einzelnen Simulatoren jeder Prozeß unabhängig von den Wartebedinungen der anderen arbeiten kann.

Diese Problematik ist bei der folgenden Konkretisierung für die einzelnen Verfahren jedoch ggf. nochmals zu diskutieren. Untersuchungen mit einer Vielzahl elementarer Prozesse unter Verwendung eines leistungsfähigen Hardware-Schedulers mit minimalem Overhead ergaben auf einem Transputercluster jedoch auch, daß der Synchronisationsaufwand der lokalen Teilmodelle untereinander nicht zu vernachlässigen ist [MMR93a].

Die Ereignisliste wird mit der Zerlegung ebenfalls verteilt und beinhaltet nur noch diejenigen

Ereignisse, die lokale Objekte des geclusterten Teilmodells betreffen. Allerdings sind Ereignisse, die für Objekte in anderen Teilen des Gesamtmodells generiert werden, an diese zu propagieren. Beim Empfänger müssen die Ereignisse, die andere Simulatoren generieren, entgegengenommen werden. Der Austausch dieser Ereignisse erfolgt durch Einplanen in die Ereignisliste des Partners, was entweder durch (geschützten) Zugriff auf gemeinsame Speicherbereiche oder durch Versenden von Ereignisnachrichten geschieht. Eine Ereignisnachricht beinhaltet in der Regel den Zeitstempel, zu dem das Ereignis generiert wurde, den Zeitpunkt, zu dem es beim Empfänger eingeplant werden soll, das zugehörige Objekt und den Typ des Ereignisses. Bei der Ankunft wird die Nachricht in ein lokales Ereignis umgewandelt und in die Ereignisliste entsprechend eingekettet.

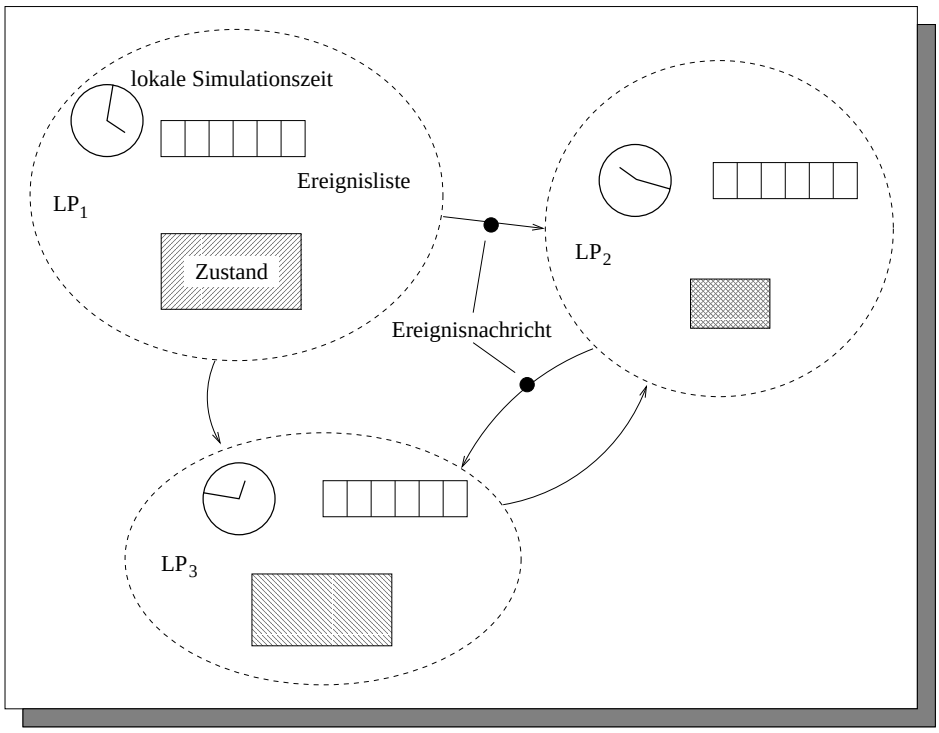


Abbildung 2.8: Kooperation sequentieller Simulatoren

Da die Ereignisse (völlig) asynchron bearbeitet werden können, gilt dasselbe auch für das Generieren neuer Ereignisse für andere Simulatoren. Im Zusammenhang mit den unterschiedlichen Werten der lokalen Uhren und dem Fehlen einer globalen Sicht auf das Gesamtsystem kann hierdurch sehr leicht die Kausalität verletzt werden. Da solche Verletzungen (verstärkt durch Nichtdeterminismen bei Nachrichtenlaufzeiten) zu verfälschten Simulationsergebnissen führen können, sind geeignete Maßnahmen zur Abwehr oder Vermeidung zu ergreifen.

Ein erster Ansatz zur Kontrolle kausaler Abhängigkeiten in verteilten Systemen stellt die Lamport-Zeit dar [LAM78a]. Hierbei werden neue Ereignisse zum Zeitpunkt ihrer Generierung mit einem Zeitstempel versehen. Diese Zeitstempel werden über ein System logischer Uhren c_i erzeugt, die eine Abbildung der Ereignisreihenfolge in eine partial geordnete Menge vornehmen. Dabei gelten zwei Regeln:

1. Sind e_1 und e_2 zwei lokale Ereignisse eines Prozessors P_i und wurde e_1 vor e_2 generiert, so gilt: $c_i(e_1) < c_i(e_2)$.

2. Der Lamport-Zeitstempel eines Empfangsereignisses ist stets größer als der des Sendeereignisses und des letzten lokalen Ereignisses beim Empfänger.

Gilt die Happened-Before-Relation $\rightarrow$ mit $e_1 \rightarrow e_2$ für kausale Ereignisse beliebiger Prozesse, dann spiegelt sich diese Relation in den Lamport-Zeitstempeln wieder, da der Zeitstempel von e_1 kleiner als der von e_2 ist. Die Wahrung aller kausalen Abhängigkeiten — auch unter Beachtung der transitiven Hülle über Prozessorgrenzen hinweg — ist das Grundproblem bei der Realisierung der parallelen ereignisgesteuerten Simulation. Prinzipiell dürfen also nur kausal unabhängige Ereignisse e_i und e_j gleichzeitig ausgeführt werden, d.h. es gilt: $e_i \not\rightarrow e_j$ und $e_j \not\rightarrow e_i$.

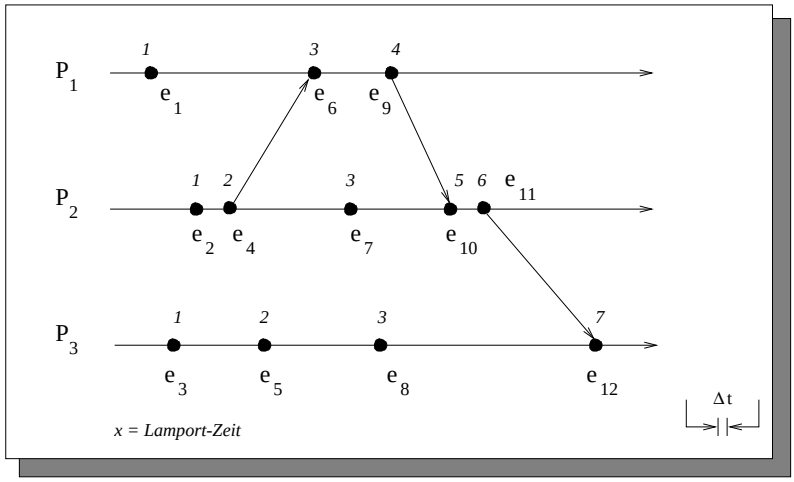


Abbildung 2.9: Parallel ausführbare Ereignisse mit Lamport-Zeit

Ein Beispiel für parallel ausführbare Ereignisse sind die Ereignismengen $E_1 = \{e_1, e_2, e_3\}$, $E_2 = \{e_4, e_5\}$, $E_3 = \{e_6, e_7, e_8\}$, $E_4 = \{e_9\}$, $E_5 = \{e_{10}\}$, $E_6 = \{e_{11}\}$, $E_7 = \{e_{12}\}$ in Abb. 2.9.

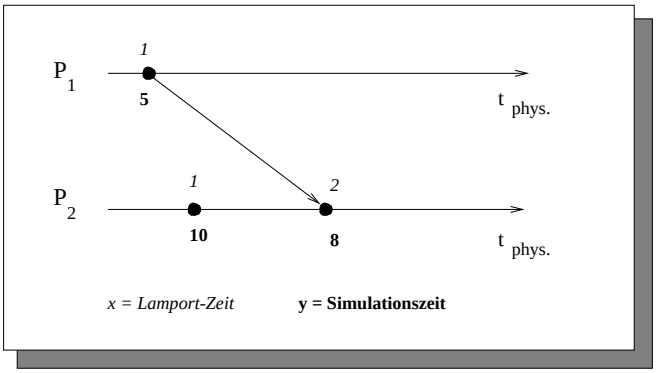


Abbildung 2.10: Unverträglichkeit von Lamport- und Simulationszeit

Hier zeigt sich das Beschleunigungspotential, das die asynchrone ereignisgesteuerte Simulation gegenüber anderen Verfahren aufweist. Bei einem synchronen Simulationsverfahren würde, ein kleines Δt vorausgesetzt, innerhalb jedes Zeitrasters höchstens ein Ereignis bearbeitet, da alle Simulationszeiten verschieden sind. Effektiv hat man bei einer ungünstigen Verteilung der Eintrittszeitpunkte der Ereignisse einen stark aufgeblähten sequentiellen Simulator. Die ereignisgesteuerte

Simulation benötigt jedoch unter der Annahme, daß alle Ereignisroutinen in etwa gleich lange Berechnungsdauer haben, nur 7 statt 12 Zeitraster.

Gleichzeitig offenbart das Beispiel aber auch eine Schwäche des Verfahrens. Die Empfangsereignisse können nicht vor den sie generierenden Sendeereignissen ausgeführt werden. Existieren nun lange Ketten von Nachrichten, ohne daß weitere lokale Ereignisse vorhanden sind, ist die Verarbeitung der Ereignisse auch hier zwangsläufig sequentiell. Das Einplanen dieser Ereignisse wird ggf. durch die Nachrichtenlaufzeiten weiter verzögert, die es im sequentiellen Simulator nicht gibt.

Leider reicht die Lamport-Zeit zur Erhaltung der kausalen Abhängigkeiten aber nicht aus, da für parallele ereignisgesteuerte Simualtion die korrekte Reihenfolge der Simulationszeitstempel und nicht die Ausführung der Ereignisse in ihrer realen Generierungsreihenfolge maßgeblich ist. Abb. 2.10 veranschaulicht diese Problematik. Die horizontalen Achsen stellen diesmal die Realzeit dar und ein Punkt ist der reale Erzeugungszeitpunkt eines Ereignisses, anhand dessen sein Lamport-Zeitstempel generiert wird.

Jefferson weist auf diesen Umstand hin, indem er das Problem der Reihenfolgeerhaltung bezüglich der Simulationszeit als invers zu Lamports logischen Uhren bezeichnet [JEF85a]. Bei Lamport geht es darum, beliebigen Ereignisse Zeitstempel, die die kausale Erzeugungsreihenfolge respektieren, zuzuteilen. In der diskreten ereignisgesteuerten Simulation existieren bereits vergleichbare Zeitstempel für die Simulationszeit. Diese geben die Kausalität und geforderte Anordnung bereits vor. In der Regel weicht die dadurch definierte Reihenfolge von der Folge der realen Generierung jedoch ab. In Abb. 2.10 führt also die Zuordnung eines Lamport-Zeitstempels zu keiner Lösung der Kausalitätsproblematik. Es kann dadurch lediglich auf die Reihenfolge, in der Ereignisse generiert wurden, aber keinesfalls auf ihre letztendliche Ausführungsreihenfolge in einem Simulationslauf geschlossen werden.

2.3.3 Weitere Vorgehensweisen

Der Vollständigkeit halber sollen noch drei weitere Parallelisierungsansätze erwähnt werden.

- Funktionale Parallelisierung [COM84a], [VEL92a] teilt die Simulationshilfsfunktionen wie Ereignisverwaltung, Eingabe- und Ausgabeoperationen, Statikstikmodule und Zufallszahlengenerierung auf mehrere Prozessoren auf. Comfort untersuchte diese Vorgehensweise am Beispiel der Schaltkreissimulation und fand wenig Beschleunigungspotential.

- Ebenso verspricht auch die in vielen Bereichen des wissenschaftlichen Rechnens eingesetzte Vektorisierung durch parallelisierende Kompiler keinen großen Erfolg [CHB83a].

- Der Einsatz von unabhängigen parallelen Simulationsläufen schließlich ist dort berechtigt, wo verschieden parametrisierte Testläufe durchgeführt werden müssen [HEI86a]. Dieses Verfahren ist jedoch trivial und nur dort hilfreich, wo viele nicht aufeinander aufbauende Tests benötigt werden.

Wir kommen nun zurück zur ausführlichen Beschreibung der Basisverfahren der PDES, die für die Realisierung der in dieser Arbeit durchgeführten Untersuchungen relevant sind. In der Literatur werden zwei grundsätzliche Verfahren zur Synchronisation der lokalen Simulatoren beschrieben, die entsprechend ihrem grundsätzlichen Vorgehen als *konservativ* bzw. *optimistisch* bezeichnet werden.

2.4 Konservative Verfahren

Die Idee der konservativen Simulationsalgorithmen geht auf nahezu zeitgleich veröffentlichte Ansätze von Bryant [BRY77a, BRY79a] und Chandy und Misra [CHM78a, CHM79a] zurück und wird in der Literatur deshalb auch als Chandy-Misra-Bryant-Verfahren (CMB) bezeichnet [SUS88a, BRI90a, SOU92a]. Weiterhin wird ein weiterer konservativer Ansatz von Peacock, Wong und Manning beschrieben [PWM79a] und *Link-time*-Algorithmus genannt.

Grundgedanke aller konservativen Ideen ist die Bemühung, Kausalitätsverletzungen in jedem Fall zu vermeiden, indem Ereignisse stets in nicht fallender Zeitstempelreihenfolge simuliert werden. Fujimoto bezeichnet diese Eigenschaft als *local causality constraint* [FUJ90a]. Misra verwendet dafür den Begriff *realizability* [MIS86a], wonach ein Simulator nur aufgrund seines lokalen Wissens agiert und bei Ausführung von Aktionen nur zurückliegende Ereignisse maßgeblich sind. Um diese Bedingungen zu garantieren, werden zwei wichtige Voraussetzungen gemacht.

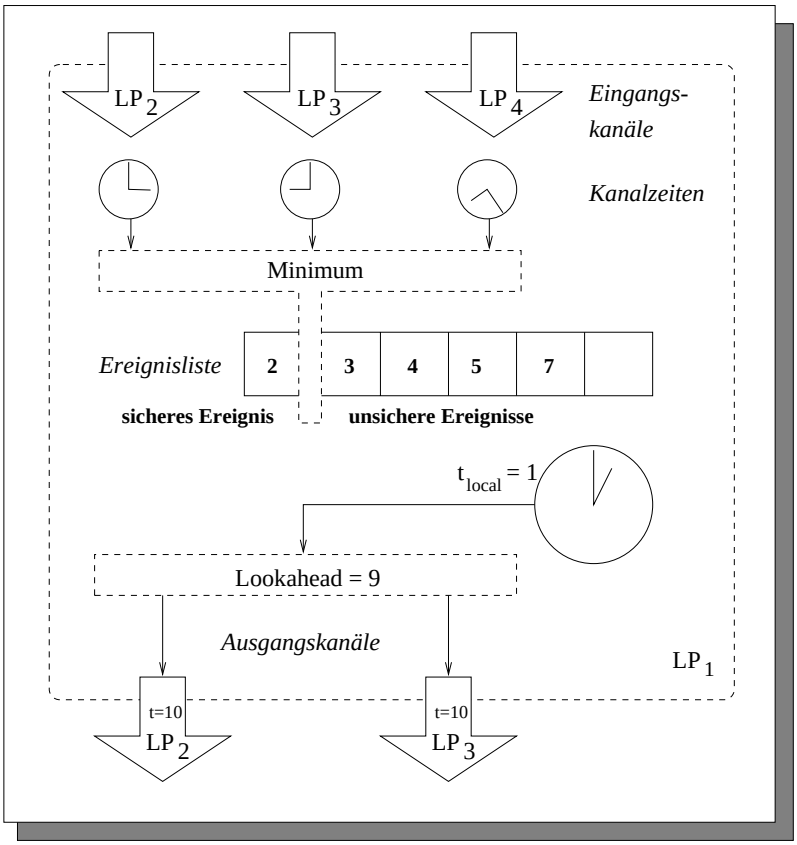


Abbildung 2.11: Struktur eines LPs bei konservativer Synchonisation

- Nachrichten werden über Kanäle verschickt, die genau einen Sender-LP mit einem Empfänger-LP verbinden. Diese Kanäle müssen die FIFO-Eigenschaft erfüllen, d.h. zwei Nachrichten zwischen einem festen Paar von LPs werden beim Ziel in der Absendereihenfolge empfangen. Für bidirektionale Kommunikation werden zwei entgegengesetzt orientierte Kanäle benötigt.

- Ein Simulator verwendet die Zeitstempel der entgegengenommenen Ereignisnachrichten als Garantien. Aufgrund einer topologischen Analyse wird vor Simulationsbeginn ermittelt, von

welchen anderen Simulatoren Nachrichten eintreffen können. Diese Kommunikationskanäle erhalten auf Empfängerseite Zeitstempel, die dem Zeitstempel der letzten empfangenen Nachricht entsprechen. Dabei wird für die Garantieberechnung im konservativen Grundalgorithmus der Ausführungszeitpunkt des in der Nachricht enthaltenen Ereignisses verwendet, der mit dem Zeitstempel des Senders übereinstimmen muß. Ein Nachrichtenkanal kann also keine Verzögerung bezüglich der Simulationszeit modellieren. Dadurch wird sichergestellt, daß alle Nachrichten eines Kanals eine nichtfallende Zeitstempelreihenfolge besitzen. Ein leerer Kanal, über den noch keine Nachricht empfangen wurde, hat den Zeitstempel des Simulationsbeginns.

Jeder Simulator führt stets nur lokale Ereignisse aus, deren Simulationszeitpunkt kleiner ist als das Minimum aller Kanalzeitstempel. Wird außerdem die (unrealistische) Annahme gemacht, daß alle Zeitstempel, die ein Simulator verarbeitet, unterschiedlich sind, so können auch Ereignisse mit dem Zeitstempel des Minimums bereits ausgeführt werden.

In Abb. 2.11 ist das Minimum der Eingangskanalzeiten 3. Alle Ereignisse mit kleinerem Zeitstempel können sicher ausgeführt werden. Das letzte Ereignis wurde zum Zeitpunkt $t_{local} = 1$ ausgeführt. Weiterhin hat der Simulator LP_1 das Wissen (Lookahead), daß er das nächste Ereignis frühestens in 9 Zeiteinheiten ausführt. Deshalb kann er ausgehende vom aktuellen Wert t_{local} seinen nachfolgenden LPs die Garantie $t = 10$ geben.

Die beiden Bedingungen garantieren, daß alle Ereignisse in einer korrekten kausalen Reihenfolge abgearbeitet werden. Die Begründung dafür liegt in folgenden Überlegungen.

- Ein Simulator, der keine eingehenden Nachrichtenkanäle hat, bearbeitet seine Ereignisse stets in aufsteigender Reihenfolge und liefert somit kausal korrekte Ergebnisse. Ereignisnachrichten, die er an andere Simulatoren verschickt, tragen somit aufsteigend sortierte Zeitstempel.
- Durch die FIFO-Eigenschaft der Kanäle bleibt die Zeitstempelreihenfolge während der Übertragung erhalten. LP_1 hat dadurch die Gewißheit, daß auf ein Ereignis, das er von LP_2 empfangen hat, nur noch Ereignisnachrichten eintreffen werden, die zu einem späteren Simulationszeitpunkt generiert wurden und keinen Einfluß mehr auf ihre Vorgänger nehmen können.
- Wird das Minimum aller Eingangskanalzeiten als Schranke für die Freigabe zur Simulation eines lokalen Ereignisses verwandt, so kann mit Sicherheit davon ausgegangen werden, daß zu einem späteren Zeitpunkt weder lokale noch globale Ereignisse generiert werden, die Einfluß auf die Vergangenheit dieses LPs nehmen könnten. Das jeweils kleinste Ereignis der Ereignisliste ist definitiv das kleinste jemals von diesem lokalen Simulator noch auszuführende Ereignis sofern sein Zeitstempel unterhalb dieser Schranke liegt.

Ausgehend von diesen Voraussetzungen ergeben sich zwei Varianten zur Realisierung konservativer Simulationsalgorithmen.

2.4.1 Deadlockvermeidung

Die Tatsache, daß ein leerer Kanal mit dem Simulationsstartzeitpunkt markiert wird, kann ebenso wie die Tatsache, daß über einen Kanal extrem wenig Nachrichten verschickt werden (weil keine dazugehörigen Ereignisse existieren), dazu führen, daß ein LP sehr lange und im schlimmsten Fall (bei einem über die gesamte Simulationszeit vollständig leeren Kanal) bis zum Simulationsende blockiert werden kann, weil er befürchten muß, daß über ihn irgendwann dennoch eine Nachricht mit kleinerem Zeitstempel als alle lokal bereits bekannten Ereignisse eintreffen könnte (Abb. 2.12).

Solche langwierigen Blockaden führen zu einem Aushungern der betroffenen logischen Prozesse. Zu ihrer Behebung schlugen sowohl Bryant [BRY79a] als auch Chandy und Misra [CHM81a] den Versand zusätzlicher Kontrollnachrichten (sog. Nullnachrichten) vor, die zum eigentlichen Simulationsvorgang keinen Beitrag liefern und an relevanter Information nur einen Zeitstempel des Absenders tragen.

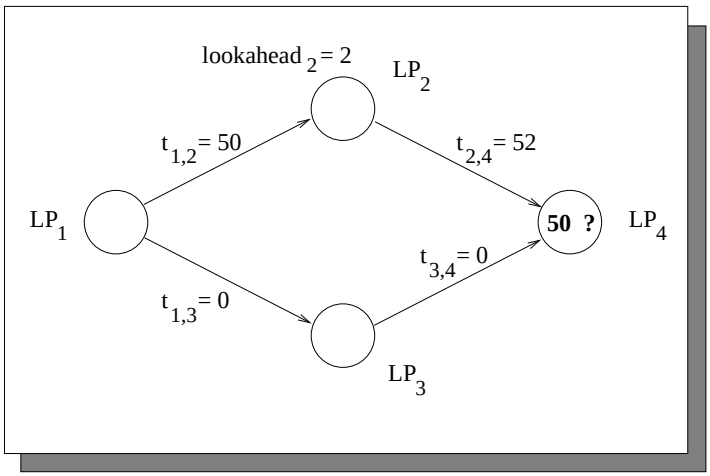


Abbildung 2.12: Aushungern durch leere Kanäle

Nach der Abarbeitung eines Ereignisses, die in der Regel eine Erhöhung der lokalen Simulationszeit bewirkt, werden neben den eigentlichen Ereignisnachrichten auch zusätzliche Nachrichten zum Propagieren der Änderung der lokalen Simulationszeit an alle LPs verschickt, zu denen der bearbeitende LP ausgehende Kanäle besitzt und die nicht ohnehin eine normale Ereignisnachricht erhalten. Anhand der in den Nullnachrichten enthaltenen Zeitstempel, die der lokalen Uhr des Senders entsprechen, lassen sich beim Empfänger ggf. Kanalzeiten erhöhen und weitere Ereignisse ausführen. Da dadurch einige Deadlocks vermieden werden können, ist das Verfahren auch unter dem Namen *Deadlock-avoidance* bekannt.

Ein wesentlicher Nachteil dieser Vorgehensweise ist jedoch die mitunter beträchtliche Erhöhung des Nachrichtenaufkommens. Um diesem Overhead zu begegnen, schlug Reynolds ein anforderungsbasiertes Verfahren mit dem Namen SRADS (Shared-resource-algorithm-for-distributed-simulation) vor [REY82a]. Hierbei kontrolliert ein Simulator in der Rolle eines Empfängers periodisch auf einer für Sender und Empfänger gemeinsam zugreiffbaren Ressource selbst, ob neue Ereignisse für ihn generiert wurden. Ist das nicht der Fall, so fordert er bei seinem Vorgänger dessen lokale Simulationszeit an, um eine Blockade zu vermeiden. Reichen die Garantien auch beim angefragten LP nicht aus, so erkundigt dieser sich wiederum bei seinen Vorgängern. Die Anzahl zusätzlicher Nachrichten wird damit stark reduziert, weil Kontrollnachrichten nur dann fließen, wenn sie wirklich benötigt werden. Peacock, Wong und Manning schlagen einen Algorithmus vor (Blocking-table) [PWM79a], der in einer Tabelle alle diejenigen Simulatoren verwaltet, die über einen mitunter indirekten Pfad mit seinen Eingängen verbunden sind. Diese werden bei fehlenden Garantien mittels einer direkten Anfrage (Request) nach ihren lokalen Simulationszeiten befragt.

Requestbasierte Verfahren müssen so erweitert werden, daß bei kaskadierten Anfragen der gesamte bisher zurückgelegte Pfad (eine Liste der besuchten LPs) in den Requests mitgeführt wird [MIS86a]. Erreicht eine Anfrage ihren Ausgangspunkt, wird der Zyklus erkannt und der Zeitstempel des kleinsten Ereignisses innerhalb des Zyklus an alle beteiligten Prozesse propagiert. Bei Misra legt die Antwort auf eine Anfrage den Weg in umgekehrter Richtung wieder zurück und aktualisiert

dabei die Zeiten der benutzten Kanäle. Derjenige LP, zu dem das minimale Ereignis gehört, darf die Simulation fortsetzen, sofern kein anderer, nicht zum Zyklus gehöriger Kanal dies verbietet.

Im Rahmen der bisher beschriebenen Algorithmen können Modelle, die Zyklen enthalten, schwerwiegende Probleme bereiten. Als einfaches Beispiel dieses Problems sollen zwei LPs betrachtet werden, die sich gegenseitig Nachrichten einplanen (Abb. 2.13). Diese Prozesse finden sich bei Verwendung des Deadlock-avoidance-Algorithmus sofort nach dem Start in einem klassischen Deadlock, weil jeder LP auf die Erhöhung der Kanalgarantien durch den anderen wartet, selbst aber aufgrund der ihm fehlenden Garantien nichts dazu beitragen kann, selbst höhere Zusagen zu machen.

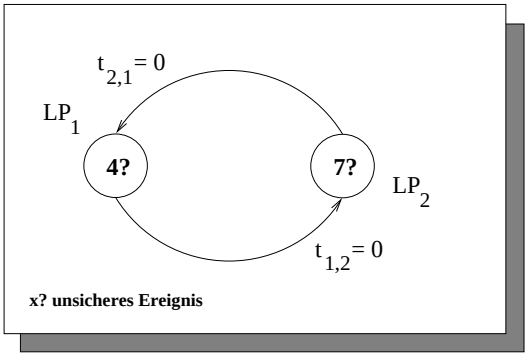


Abbildung 2.13: Grundmodell eines zyklischen Deadlocks

Ein erster Ansatz zum Aufbrechen solcher Deadlocks kann die Verwendung zusätzlichen applikationsspezifischen Wissens sein. Vermag ein LP festzustellen, wann er das nächste lokale Ereignis frühestens generieren bzw. ausführen wird, so besitzt er mit dieser Vorausschau (*Lookahead*) ebenfalls die Information, wann er frühestens Ereignisse für andere LPs erzeugen wird. In einem Warteschlangennetz mit FCFS-Stationen läßt sich diese Vorausschau bei der Ankunft jedes Jobs direkt berechnen. Der Lookahead wird bei einem Nullnachrichten-Verfahren mit den Nachrichten oder bei Requestmethoden mit der Antwort auf eine Anfrage kombiniert, um die aus diesen Werten gewonnenen zusätzlichen Garantien zu propagieren.

Zyklen lassen sich somit dann aufbrechen, wenn wenigstens ein beteiligter Simulator einen von Null verschiedenen Lookahead besitzt. Für Simulationsmodelle, die diese Eigenschaft nicht garantieren (z.B. Warteschlangen mit normalverteilter Bedienrate), ist Deadlock-avoidance mit Nullnachrichten also nicht generell einsetzbar. Existiert jedoch bei den beteiligten LPs Lookahead, so ist ein Fortschritt der Simulationszeit innerhalb eines Zyklus garantiert. Ist die Summe aller Lookaheads innerhalb eines Zyklus sehr klein, so kann durch Requests oder Nullnachrichten jedoch immer noch ein nicht unerheblicher Aufwand entstehen. Verfügt im schlechtesten Fall nur ein Prozessor über Lookahead, so verhalten sich die LPs im Zyklus nahezu wie bei zeitgesteuerter Simulation, da sich quasi alle LPs zunächst über Nullnachrichten synchronisieren, bevor ein (oder bei gleichen Zeitstempeln vielleicht auch mehrere) Ereignisse ausgeführt werden können.

Bedauerlicherweise gelten diese Überlegungen auch für Kanäle, über die lange keine Nachrichten verschickt werden. Dadurch entstehen bei parallelen, konvergenten Ästen in einem gerichteten Graphen von LPs erhebliche Verzögerungen bei der Zeitfortschaltung.

2.4.2 Deadlockerkennung und -auflösung

Eine andere Vorgehensweise läßt das Entstehen von Deadlocks grundsätzlich erst einmal zu und sieht Maßnahmen vor, diese Deadlocks zu erkennen und durch geeignete Vorkehrungen die blockierten Prozesse aus der Verklemmung zu befreien. Verfahren zur Auflösungen von Deadlocks in verteilten Simulationen wurden u.a. in [BRY79a, CHM81a, MIS86a] beschrieben. Jedoch werden von diesen Algorithmen prinzipiell nur globale Deadlocks, an denen also alle LPs beteiligt sind, entdeckt. Die Zyklenauflösung bei [PWM79a] und [REY82a] realisiert teilweise auch die Auflösung lokaler Deadlocks innerhalb der Zyklen. Die Erkennung von Deadlocks beruht bei den vorgeschlagenen Methoden entweder auf einem zentralen Kontrollprozeß [DIS80a] oder dem Verfahren von Chandy und Lamport [CHL85a] zur Berechnung verteilter Schnappschüsse.

Ein System, das (teilweise) in einen Deadlock gerät, kann sich nicht selbst daraus befreien. Alle LPs, die blockieren, befinden sich in einem passiven Zustand, aus dem sie nur durch einen anderen Prozeß (z.B. durch Erhalt einer Nachricht, die ein Ereignis oder eine Garantie enthält) wieder in den aktiven Zustand versetzt werden können. Befinden sich nur einige oder alle Prozesse im passiven Zustand und erhalten sie keine Nachrichten mehr von anderen Prozessen, so befindet sich das System in einem lokalen bzw. globalen Deadlock.

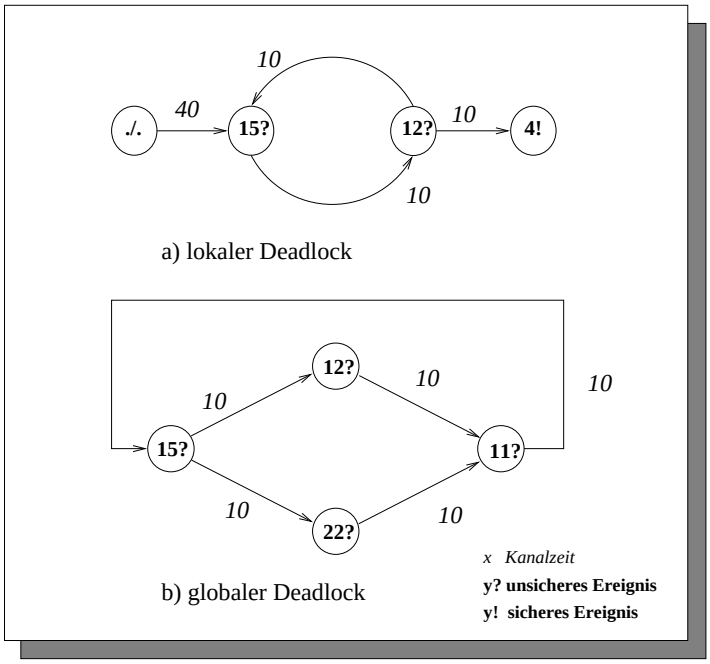


Abbildung 2.14: Globaler und lokaler Deadlock

Weitere Ansätze zur Deadlockerkennung in verteilten Systemen und dem allgemeineren Problem der Generierung von Schnappschüssen finden sich in [MAT89d]. Prinzipiell lassen sich zur Entdeckung eines Deadlocks alle Algorithmen einsetzen, die eine verteilte Terminierungserkennung durchführen [MAT87a]. Die Erkennung von Terminierung in verteilten Systemen ist jedoch nicht trivial. Insbesondere die Tatsache, daß es keine globale Sicht auf einen konsistenten Zustand eines verteilten Systems gibt, ist sehr problematisch. Außerdem müssen bei der Berechnung neben den direkten Informationen über die lokalen Zustände der Prozesse, deren Ensemble den gemeinsamen Zustand zu einem virtuell gleichzeitigen Zeitpunkt approximiert, auch die Nachrichten, die während der Beobachtung noch auf Kanälen unterwegs sind und die Teile der Zustandsinformationen bein-

halten, berücksichtigt werden.

Chandy und Misra schlagen zur Deadlockerkennung einen zentralen Kontrollprozeß vor, der nach dem Terminierungserkennungsverfahren von Dijkstra und Scholten [DIS80a] arbeitet. Prinzipiell handelt es sich dabei um ein Kreditverfahren, das auf einem Spannbaum mit dem zentralen Prozeß als Wurzel arbeitet. Mattern stellte einen darauf basierenden Algorithmus vor, der jedoch die Berechnung effizienter gestaltet und in dem ein beliebiger an der verteilten Berechnung beteiligter Prozeß die Rolle der Wurzel übernehmen und die Erkennung initiieren kann [MAT89f].

Chandy und Misra führen auch die Situation des Aushungerns einzelner LPs aufgrund leerer Paralleläste auf eine Deadlocksituation zurück. Jedoch beruht ihre Sichtweise auf dem zugrundeliegenden synchronen Kommunikationsmodell. Eine Nachricht kann im beschriebenen Ansatz nur dann verschickt werden, wenn der Kommunikationspartner zum Empfang bereit ist. Das Kommunikationsschema folgt dem von Hoare entwickelten CSP (Communicating-sequential-processes) [HOA78a] mit einer limitierten Anzahl von Pufferplätzen für Nachrichten jedes Kommunikationskanals. In diesem Ansatz kann es auch zu Deadlocks aufgrund des Kommunikationssystems kommen.

In vielen Fällen spiegeln Deadlocks nicht das Verhalten des zugrundeliegenden Modells wider, sondern sind durch Modellierung und Implementierungsverfahren für das Simulationsmodell begründet. In den realen Systemen (und sequentiellen Simulatoren) treten meist gar keine Deadlocks auf. Erst durch fehlerhafte Synchronisations- oder Kommunikationsmechanismen oder aufgrund bei der Verteilung eingeführter (zyklischer) Abhängigkeiten werden Möglichkeiten zur Verklemmung generiert, die sich nicht immer vermeiden lassen. Eine sehr sorgfältige Planung und Umsetzung ist deshalb bei der Realisierung paralleler Simulationsverfahren von Nöten. Das gilt natürlich ganz allgemein für alle parallelen Algorithen und ist insofern nicht nur für die verteilte Simulation symptomatisch.

2.5 Optimistische Verfahren

Sämtliche optimistischen Verfahren der PDES stellen im Grunde Variationen und Optimierungen des von Jefferson und Sowizral entwickelten Time Warp dar [JES85a, JEF85a]. Grundannahme der optimistischen Algorithmen ist, daß Blockaden, die in konservativen Verfahren zur Kausalitätswahrung eingesetzt werden, sich im Nachhinein oft als unnötig herausstellen. In dieser Zeit könnte, unter der Annahme, daß keine weiteren Ereignisse generiert sondern lediglich auf eine Erhöhung von Garantien gewartet wird und folglich auch keine Kausalitätsverletzung erfolgt, sinnvolle Arbeit zum Simulationsfortschritt geleistet werden. Für die seltenen Fälle, in denen dennoch vergangene Ereignisse nachträglich auftreten, werden entsprechende Vorkehrungen zum Zurücksetzen und Wiederholen der Simulation getroffen.

2.5.1 Arbeitsweise des Time Warp

Im Time Warp werden alle Ereignisse von einem LP sofort in ihrer lokalen Ereignisreihenfolge ausgeführt. Dabei werden in regelmäßigen Abständen – der Grundalgorithmus schlägt dies nach jeder Ereignisausführung vor – Sicherungen des lokalen Zustands (*Checkpoints*) erzeugt, um notfalls, d.h. bei Eintreffen einer Nachricht in der Vergangenheit (vor der lokalen virtuellen Zeit, LVT), auf einem früheren konsistenten Zustand unmittelbar vor dem Eintrittszeitpunkt des verspäteten Ereignisses (*Straggler*) aufsetzen zu können. Dieses Zurücksetzen gleicht dem Zurück- oder Aufrollen der gespeicherten Zustandsfolge entgegen der Zeitachse und wird als *Rollback* bezeichnet. Sicherungsverfahren werden u.a. auch in langlaufenden sequentiellen Programmen verwendet, um im Fall eines Programmfehlers oder Rechnerabsturzes eine Berechnung nicht vollständig neu durchführen

zu müssen. In Datenbanken sind bei Sperrverfahren ebenfalls Rücksetzaktionen mit Aufsetzen auf einem konsistenten Zustand bei Transaktionsabbrüchen z.B. aufgrund von Konflikten bei der Akquisition von Sperren anzutreffen. Die Abarbeitung von de fakto noch unsicheren Ereignissen in der Simulation entspricht damit in etwa dem Start einer Transaktion unter der Annahme, man werde alle benötigten Sperren erhalten.

Zwangsläufig generieren in einem verteilten Simulator die einzelnen LPs auch Ereignisnachrichten, die an die anderen LPs verschickt werden. Im Falle eines Rollbacks müssen deshalb auch die extern ausgelösten Ereignisse widerrufen werden. Zu diesem Zweck erhalten alle Nachrichten ein zusätzliches Kennzeichen, das konventionelle Ereignisnachrichten mit einem "+" als positive Nachrichten markiert. Die Auswirkungen einer positiven Nachricht werden durch Versenden eine identischen Nachricht allerdings mit negativer Markierung "-" widerrufen. Ähnlich dem Konzept von Materie-Antimaterie-Auslöschung heben sich die korrespondierenden Ereignisnachrichten mit unterschiedlichen Vorzeichen beim Aufeinandertreffen auf. Als Teil der Zustandssicherung werden deshalb neben den Werten der lokalen Zustandsvariablen auch alle Nachrichten, die verschickt werden, als sogenannte Antinachrichten (mit negativem Vorzeichen) nach ihrem *Absendezeitpunkt* sortiert in einer Ausgangsliste (Output-Queue) gespeichert. Ebenso verwaltet ein Empfänger alle erhaltenen Nachrichten nach *Empfangszeitstempel* sortiert in einer Input-Queue (Abb. 2.15).

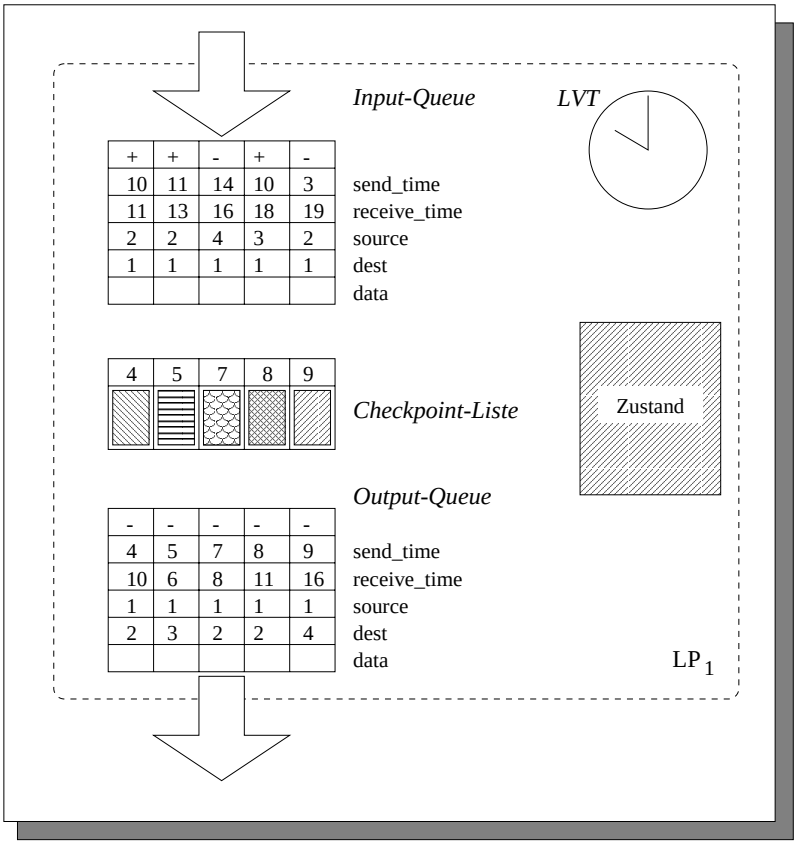


Abbildung 2.15: Struktur eines LPs im Time Warp

Die Kommunikation zwischen zwei LPs muß bei Time Warp nicht mehr dem FIFO-Prinzip entsprechen. Deshalb kann es geschehen, daß eine Antinachricht vor der zugehörigen positiven Nachricht beim Ziel eintrifft. Sie wird ebenso wie gewöhnliche Nachrichten bei der Ankunft gespei-

chert. Allerdings werden für Antinachrichten keine Ereignisse in die Ereignisliste eingetragen. Trifft später die zugehörige Nachricht ein, so löschen sich beide gegenseitig aus, ohne daß es weiterer Aktionen bedarf. Dasselbe gilt, wenn die Ereignisnachricht bereits in der Input-Queue vorhanden ist, aber das zugehörige Ereignis noch nicht abgearbeitet wurde. Beide Nachrichten und das Ereignis werden entfernt.

Der eigentlich kritische Fall liegt vor, wenn ein Ereignis bereits bearbeitet ist und anschließend durch eine Antinachricht widerrufen wird. Dann wird (wie bei einem verspäteten Ereignis) ein sekundärer Rollback beim Ziel-LP ausgeführt, der wiederum durch Versenden weiterer Antinachrichten zusätzliche Rollbacks verursachen kann. Das von Jefferson beschriebene Verfahren garantiert jedoch, daß die Tiefe aller Rollbacks durch den Zeitstempel des kleinsten noch nicht bearbeiteten Ereignisses im System begrenzt wird [JEF85a]. Dieser Zeitpunkt wird auch als *Global-virtual-time (GVT)* bezeichnet. Die GVT beschreibt den Gesamtfortschritt der Simulation und ist eine monoton wachsende Größe. Zu ihrer Berechnung müssen wie bei Deadlockerkennungsalgorithmen sowohl lokal vorhandene Ereignisse als auch Nachrichten, die noch unterwegs (*intransit*) sind, berücksichtigt werden.

Die Freiheit, einen LP nicht blockieren zu müssen, wird in optimistischen Verfahren somit durch zusätzlichen Speicherplatzbedarf erkauft. Dieser Bedarf kann bei größeren Modellen dramatisch steigen, so daß geeignete Vorkehrungen zur Freigabe nicht mehr benötigter Sicherungskopien von Zuständen und Nachrichten benötigt werden. Da ein Rollback nicht hinter die GVT zurückführen kann, werden alle Sicherungsdaten mit kleineren Zeitstempel nicht mehr benötigt. Allerdings muß stets der Checkpoint, der unmittelbar vor dem aktuellen Wert der GVT erstellt wurde, aufbewahrt werden, da ein Rollback auch exakt auf die GVT zurückführen kann. Nachfolgend werden Aspekte bei der GVT-Berechnung sowie Varianten des Time Warp beschrieben.

2.5.2 GVT-Berechnungen

Zur Berechnung der GVT können beliebige Algorithmen, die einen verteilten Schnappschuß generieren oder verteilte Terminierung erkennen, eingesetzt werden [MMS91a]. Allerdings wird in den meisten Fällen nicht der exakte Wert der GVT sondern eine Approximation ermittelt, die analog zur echten GVT-Funktion monoton wächst und eine untere Schranke für den realen Wert darstellt (Abb. 2.16). Ziel eines guten GVT-Algorithmus ist, die Differenz zwischen ermitteltem und realem Wert gering zu halten, damit die Freigabe nicht mehr benötigter Sicherungsdaten so schnell wie möglich erfolgen kann.

Dieses Hinterherhinken des berechneten Werts ist dadurch begründet, daß zur Berechnung eines verteilten Schnappschusses Zeit verbraucht wird. Die Initiierung des Schnappschusses und das Einsammeln der lokalen Zustände der einzelnen Prozessoren erfolgt über Nachrichten. Da die eigentliche Simulation während der GVT-Berechnung nicht angehalten werden soll, sind die ermittelten Daten bei ihrem Eintreffen im Koordinatorprozeß ggf. bereits veraltet. Der ermittelte Wert repräsentiert somit lediglich den Wert der GVT zu einem virtuell gleichzeitigen Punkt auf der Realzeitachse, der zwischen Start- und Stoppzeitpunkt der GVT-Berechnung liegen kann (Abb. 2.17). Bei Bekanntgabe eines neuen Approximationswerts der GVT ist die reale GVT in der Regel bereits weiter fortgeschritten. Samadi [SAM85a], Bauer und Sporrer [BSK92a] sowie Aji, Palaniswamy und Wilsey [APW93a] beschreiben Verfahren, die über einen zentralen Kontrollprozeß die Berechnung koordinieren. Dezentrale Berechnungsansätze finden sich in [CHL85a] und [MAT89d, MAT93a].

Auf dem Basisalgorithmus des Time Warp setzen viele Modifikationen auf, die eine Optimierung des Grundverfahrens zum Ziel haben. Dabei läßt sich nach Optimierung der Laufzeit und Beschränkung der Speicheranforderungen unterscheiden, wobei die einzelnen Verfahren nicht unbe-

dingt eindeutig einem der beiden Bereiche zugeordnet werden können. Außerdem sind einige Verfahren speziell auf das zugrundeliegende Kommunikationsmodell abgestimmt (gemeinsamer Speicher oder nachrichtenbasiert).

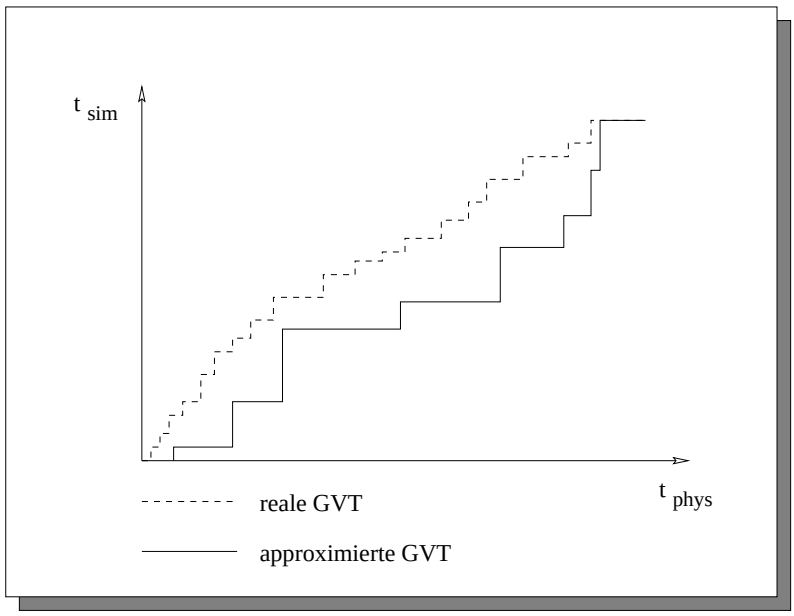


Abbildung 2.16: Approximation der GVT

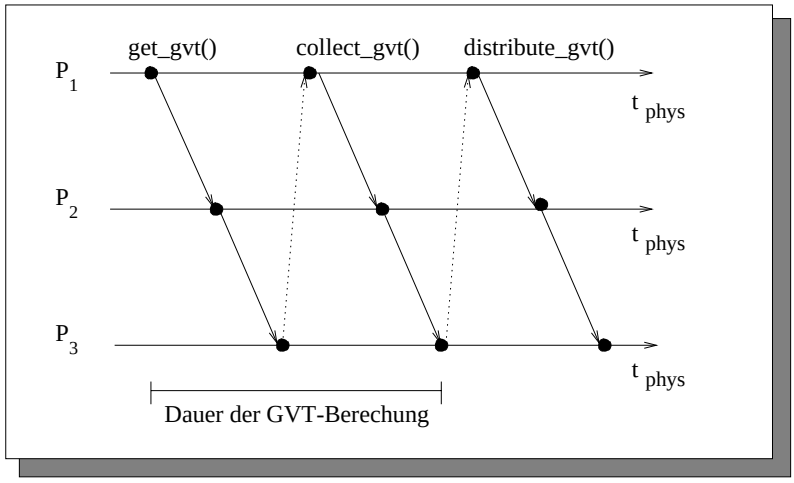


Abbildung 2.17: Rechenzeitbedarf für GVT-Berechnung

2.5.3 Lazy-cancellation

Im Basisverfahren der optimistischen Simulation werden bei einem Rollback sofort alle zu verschickenden Antinachrichten an ihre Empfänger gesandt. Diese Methode zur Auslöschung externer Effekte einer möglicherweise fehlerhaften Berechnung wird deshalb auch als *Aggressive-cancellation* bezeichnet. Wird jedoch nach dem Rollback wieder eine identische Nachrichtenfolge generiert, so

stellt sich diese Aggressivität als übereilt und überflüssig heraus[3]. Gafni schlug deshalb das Verfahren der *Lazy-cancellation* vor [GAF88a], das bei einem Rollback die entsprechenden Antinachrichten nur als potentielle Kandidaten für einen späteren Versand markiert. Bei der Generierung neuer Nachrichten im Anschluß an einen Rollback wird zunächst getestet, ob eine bis auf das Vorzeichen identische, markierte Nachricht existiert, weil die neue Nachricht vor einem Rollback bereits einmal an den Empfänger verschickt wurde. In diesem Fall ist die neue Nachricht überflüssig und kann weggeworfen werden. Außerdem wird die Markierung der Antinachricht entfernt. Findet sich in der Output-Queue keine passende Antinachricht, so wird die neu generierte Ereignisnachricht ganz normal verschickt.

Gleichzeitig wird nach der Abarbeitung von Ereignissen die Output-Queue dahingehend untersucht, ob markierte Antinachrichten mit Zeitstempeln, die kleiner als die lokale Simulationszeit sind, existieren. Dies sind die effektiv zu versendenden Antinachrichten, da ihre zugehörigen Ereignisse im korrigierten Simulationslauf nicht wieder generiert wurden. Das Versenden von Antinachrichten erfolgt bei Lazy-cancellation also mitunter wesentlich später als bei der aggressiven Variante. Die Reduktion des Nachrichtenaufwands ist somit gegen die Folgen eines verspäteten Widerrufs fehlerhafter Ereignisse abzuwägen. Tendenziell zeigen Untersuchungen, daß sich Lazy-cancellation im allgemeinen bezüglich der Laufzeiten besser verhält als die aggressive Version [LCU88a], obwohl sich Szenarien finden lassen, in denen sich jedes Verfahren nahezu beliebig schlecht im Vergleich zum anderen verhält [RFB90a, JER91a, GUN94a].

2.5.4 Lazy-reevaluation

Einen ähnlichen, allerdings auf dem Vergleich von Zuständen beruhenden Ansatz stellt *Lazy-reevaluation* dar [WES88a]. Dabei wird vor der Ausführung des Rollbacks ein Vergleichszustand berechnet, der das verspätete Ereignis (Straggler) bzw. die Antinachricht berücksichtigt. Stimmen die Zustände ohne und mit Ausführung der Korrektur überein, so hat dieses Ereignis keinen direkten Einfluß auf den lokalen Zustand. Die Ausführung eines Rollbacks ist deshalb nicht nötig. Der Vergleich von Zuständen kann jedoch auch hohe Komplexität haben und erfordert entsprechende Unterstützung durch den Programmierer des Simulationsmodells. Diese Vorgehensweise erschwert wiederum die einfache Verwendung eines möglichst generischen verteilten Simulators. Palaniswamy, Aji und Wilsey entwickelten einen Schaltkreissimulator mit Lazy-reevaluation [PAW92a] und als weitere Variante *Rollback-relaxation*, bei dem anhand einer Analyse vor Simulationsbeginn festgelegt wird, welche Ereignistypen, bei welchen Objekten Zustandsänderungen bewirken. Danach wird entschieden, wann eine Sicherung des Zustands und Rollbacks überhaupt nötig sind. Ähnliche auf dem Vergleich von Zuständen und der Kategorisierung nach zustandsändernden und nicht ändernden Ereignissen beruhende Verfahren finden sich bei [SBW88a, WLB88a].

2.5.5 Optimierung des Widerrufs von Antinachrichten

Um den Nachteil der verspäteten Revokation fehlerhafter Antinachrichten wenigstens teilweise zu kompensieren, sollten Antinachrichten nach Möglichkeit höher priorisiert werden, so daß sie bevorzugt vor anderen Nachrichtentypen behandelt werden. Dies setzt natürlich eine entsprechende Unterstützung durch das Kommunikationssystem voraus. Zwei weitere Verfahren, die Divergenz der LPs bezüglich der Simulationszeiten einzudämmen, sind die sogenannten *Wolf-calls* und *Direct-cancellation*. Ein Wolf-call ist eine spezielle Kontrollnachricht, mit der diejenigen Zielprozesse, die

[3]Dieser Fall kann Auftreten, wenn ein Ereignis keine relevanten Zustandsänderungen auslöst wie z.B. der Signalwechsel eines Eingangs am OR-Gatter, wenn der zweite Eingang permanent auf "1" liegt.

innerhalb einer gewissen Umgebung des Initiators liegen, generell ohne Spezifikation konkreter Antinachrichten über möglicherweise falsche Berechnungen des Absenders informiert werden, um so die Berechnungen der Zielprozessoren zu verzögern, indem ein Rollback auf die LVT des Initiators durchgeführt wird. Dadurch soll die Fortpflanzung der fehlerhaften Ergebnisse gebremst werden [MWM88a].

Direct-cancellation ist in Shared-memory-Systemen anwendbar und löscht fehlerhafte Ereignisse über einen in der Output-Queue gespeicherten Zeiger auf die zugehörige Nachricht im Speicher des Zielsimulators [FUJ89d].

Weiterhin kann durch Kombination mehrerer Antinachrichten zu einer längeren Nachricht (Packen) zum einen die Gesamtanzahl gesendeter Nachrichten gesenkt und außerdem alle Antinachrichten, die vom selben Rollback ausgelöst werden, auch auf dem Zielsimulator als ein einziger Rollback abgearbeitet werden. Matsumoto und Taki schlagen für FIFO-Kanäle den Versand der ältesten Antinachricht vor, die implizit alle späteren Nachrichten desselben Absenders beim Ziel ebenfalls löscht [MAT92a].

2.5.6 Speicherbedarf

Das Ziel der bisher geschilderten Modifikation des Time Warp ist im wesentlichen die Optimierung der Laufzeit durch Reduktion spezifischen Overheads, der durch die verteilte Programmierung und damit notwendige Synchronisation eingeführt wurde.

Ein ebenfalls nicht zu vernachlässigender Gesichtspunkt ist die Kontrolle des Speicherbedarfs einer verteilten Applikation. Durch die Verteilung des Modells auf mehrere Prozessoren wird es manchmal erst möglich, Modelle mit einem für einen einzelnen Rechner zu hohen Speicheraufwand zu simulieren. Dennoch können gerade die zusätzlichen Verwaltungsstrukturen und das starke Anwachsen von Sicherungsdaten im Time Warp aufgrund des Auseinandertriftens der lokalen Uhren aller LPs den durch die Verteilung zusätzlich verfügbare Speicher ausschließlich hierfür aufbrauchen.

Ein intelligentes Speichermanagement ist deshalb für Time Warp-Verfahren nötig. Sokol, Briscoe und Wieland schlagen eine Beschränkung der zur Simulation freigegebenen Ereignisse durch ein sich am Fortschritt der GVT orientierendes Simulationszeitfenster fester Größe vor [SBW88a]. Alle Ereignisse innerhalb dieses Moving-time-windows (MTW), das für alle LPs denselben festen Wert besitzt, dürfen ausgeführt werden. Existieren keine solchen Ereignisse mehr, so blockiert der zugehörige LP, bis sich das Fenster (durch Erhöhung der GVT) weiterbewegt hat. Aufgrund der zeitlichen Limitation laufen die lokalen Uhren der Simulatoren nicht mehr so stark auseinander, so daß der Speicherbedarf der Verwaltungsstrukturen ebenfalls, abhängig von der Fenstergröße, begrenzt bleibt.

Für Extremfälle, in denen ein LP beim Empfangsversuch einer neuen Nachricht dennoch an die Grenze der verfügbaren Speicherkapazität gelangt, schlägt Gafni vor, eine beliebige andere Ereignisnachricht mit größerem Zeitstempel als das neue Ereignis an ihren Absender zurückzuschicken [GAF85b]. Die Auswahl der Nachricht ist dabei beliebig. Dadurch kann der Empfänger die Nachricht mit kleinerem Zeitstempel nun annehmen und verarbeiten. Die zurückgeschickte Nachricht löst bei ihrem ursprünglichen Sender ggf. ebenfalls einen Rollback aus und löscht dabei ihr Original. Da es sich um keinen Straggler im eigentlichen Sinne handelt, wird die Nachricht später erneut generiert und wieder versendet. Zu diesem Zeitpunkt wird erwartet, daß der Empfänger aufgrund fortgeschrittener GVT und durchgeführter Garbage-collection oder aufgrund anderer Rollbacks dann wieder über freien Speicher zu ihrer Bearbeitung verfügt.

Eine Optimierung von Gafnis Verfahren stellt Jeffersons *Cancelback-Protokoll* dar [JEF90a]. Bei Speicherknappheit wird unabhängig von der verursachenden Komponente (Checkpointing, Input-

oder Output-Queue) eine beliebige der folgenden Aktionen ausgelöst. Dabei kann

- eine beliebige Nachricht der Input-Queue an den Absender zurückgeschickt, oder
- eine beliebige Antinachricht der Output-Queue verschickt, oder
- ein lokaler Zustand gelöscht werden, woraufhin in der Regel ein Rollback ausgeführt wird, sofern Checkpoints nach jedem Ereignis benötigt werden.

Alle Alternativen sorgen dafür, daß lokal und auch auf den anderen involvierten Prozessoren Speicher freigegeben wird und somit eine drohende Verklemmung der LPs vermieden werden kann. Außerdem garantiert das Verfahren, daß die Simulation, wenn auch unter starken zeitlichen Verzögerungen, überhaupt beendet werden kann und im Extremfall sogar nicht mehr Speicherplatz benötigt als ein sequentieller Simulator (der, wie bereits erwähnt, aber bei großen Modellen ggf. auf einem einzigen Rechner nicht ausgeführt werden kann). Ein konservatives Verfahren kann dagen mitunter ein (konstantes) Vielfaches an Speicher benötigen [LLB89b, JEF90a]. Die Speicherminimalität des Time Warp wurde insbesondere auf Shared-memory-Systemen beobachtet [FUJ89b]. Voraussetzung ist jedoch, daß der Nachrichtenversand atomar mit Nachrichtenlaufzeiten in Null-Zeit erfolgt und die GVT stets aktuell und ebenfalls atomar zugreifbar ist. Auf Architekturen mit verteiltem Speicher sind aus diesen Gründen einige Abstriche hinsichtlich der Speicherminimalität zu machen. Außerdem kann von einer zeitlichen Optimierung in beiden Fällen keine Rede mehr sein.

2.5.7 Zustandssicherung

Schließlich existieren auch noch verschiedene Varianten bezüglich der Sicherung der Zustandsvariablen, mit deren Hilfe eine Reduktion des Speicherbedarfs erreicht werden soll. Lin und Lazowska schlagen eine periodische Sicherung des Zustands in einem festen, größeren Abstand als nach jeder Ereignisabarbeitung vor [LIL89a]. Dieses Verfahren wird mit erhöhten Kosten für Rollbacks erkauft, da nicht mehr nach jedem Event aktuelle Zustandsdaten vorliegen. Es muß somit auf einen älteren Checkpoint zurückgegriffen werden, wodurch die Tiefe des Rollbacks und die Anzahl der zurückgesetzten und nochmals auszuführenden Ereignisse erhöht wird. Die Autoren entwickelten zur Ermittlung eines optimalen Abstands eine Formel, die auf verschiedenen Parametern wie mittlere Rollbacktiefe und Aufwand für die Erstellung eines Checkpoints, sowie Nachrichtenaufkommen und Kosten eines Simulationsschritts beruht. In [APW93a] wird aufgrund empirischer Untersuchungen an Schaltkreissimulationen ein Intervall von $7 - 10$ Ereignissen als optimale Größe für das Checkpointing ermittelt.

Eine Alternative zum periodischen Checkpointing stellt die inkrementelle Zustandssicherung dar, die lediglich die von einem Ereignis ausgelösten Änderungen protokolliert. Sie empfiehlt sich, wenn Modifikationen anhand weniger Zustandsvariablen leicht zu identifizieren und nicht umfangreich sind. Es werden ähnlich wie bei Datenbanken Undo-Informationen zu den abgearbeiteten Ereignissen gespeichert. Allerdings erhöht sich auch hier der Aufwand für Rollbackbearbeitung, da alle Ereignisauswirkungen quasi schrittweise wieder rückgängig gemacht werden müssen (Abb. 2.18). Für die Schaltkreissimulation beschreiben Bauer, Sporrer und Krodel [BSK92a, BAS93a] ein elegantes Verfahren, das die inkrementelle Zustandssicherung in die Ereignislistenverwaltung integriert. Dazu speichern die abgearbeiteten Ereignisse die Informationen zum Rückgängigmachen ihrer Auswirkungen in zusätzlichen Einträgen. In der Regel handelt es sich dabei um die Werte der betroffenen Zustandsgrößen vor der Ereignisausführung.

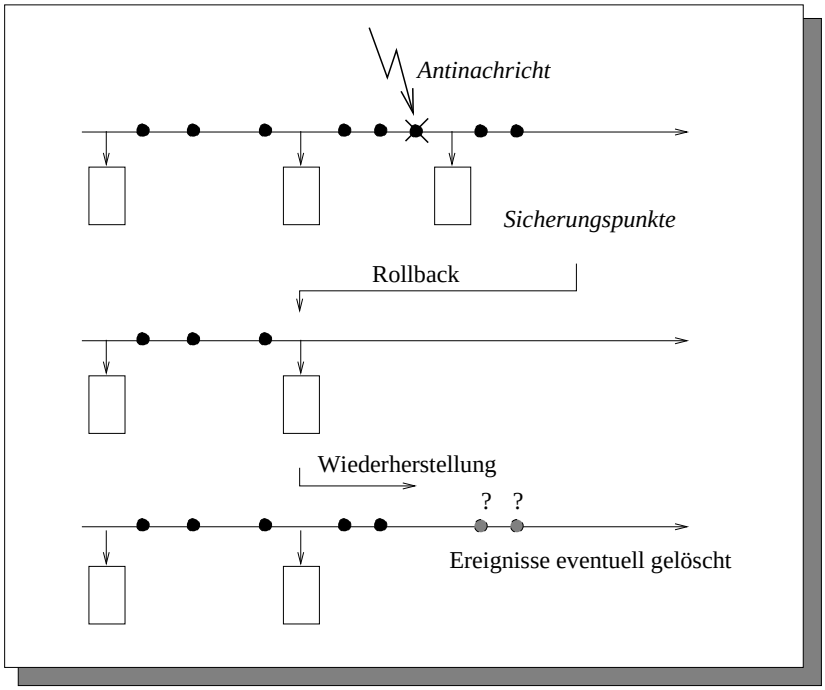


Abbildung 2.18: Rollbacks bei periodischer Zustandssicherung

2.6 Weiterentwickelte Verfahren

2.6.1 Space-time-Simulationen

Chandy und Sherman erweiterten die bisher vorgestellten optimistischen und konservativen Verfahren um die Zeit als weitere Verteilungskomponente eines Simulationsmodells [CHS89b]. Während in den bisherigen Ansätzen eine rein räumliche Aufteilung der LPs stattfand, schlagen die Autoren vor, einen einzelnen LP, der bei Chandy und Sherman als Simulationsprozeß (SP) bezeichnet wird, auch für verschiedene disjunkte Zeiträume unterschiedlichen Prozessoren zuzuordnen. Die Simulation kann somit auch für verschiedene Epochen parallel erfolgen. Der Startzustand der einzelnen SPs mit unterschiedlichen Startzeitpunkten wird bei Simulationsbeginn entweder vom Entwickler des Modells spezifiziert, aus dem Modell abgeleitet oder einfach geraten (quasi beliebig mit einigermaßen plausiblen Werten initialisiert).

Nach Beendigung der Simulation einer Epoche teilt jeder SP seinen direkten zeitlichen Nachfolgern den Endzustand der Berechnungen der bearbeiteten Objekte mit, der, ausgehend von den eigenen Belegungen beim Start, der real berechnete Startzustand des Nachfolgers ist. Unterscheidet sich der übermittelte Zustand vom bisherigen Startzustand des Nachfolgers, so wiederholt dieser die Simulation auf Basis der neuen Informationen. Die Autoren gehen davon aus, daß die Startzustände an den Schnittstellen der SPs mit der Zeit gegen Fixpunkte konvergieren und bei Erreichen eines stabilen Zustands das Simulationsende erreicht wird. Durch gemeinsame Nutzung räumlicher und zeitlicher Parallelität kann somit weiteres Beschleunigungspotential gegenüber den rein räumlichen Parallelisierungsverfahren genutzt werden. Die räumlichen Verfahren lassen sich jedoch ebenso wie die sequentielle und die parallele zeitgesteuerte Vorgehensweise durch das Space-time-Verfahren beschreiben.

2.6.2 Hybride Methoden

Unter hybriden Methoden versteht man die Kombination der reinen optimistischen und konservativen Verfahren, um deren Vorteile zu vereinbaren und die Nachteile zu vermeiden. Bei den konservativen Verfahren spart man sich den Aufwand zur Protokollierung vergangener Zustände, während bei optimistischer Vorgehensweise keine Blockaden entstehen können. Reynolds entwickelte eine Klassifizierung von Simulationsverfahren, die die Spezifikation einer Vielzahl hybrider Verfahren gestattet [REY88a]. Anhand von zehn Entwurfsvariablen lassen sich alle parallelen Simulationsverfahren weitgehend exakt beschreiben. Die beiden wichtigsten Größen sind dabei Risiko (*risk*) und Agressivität (*aggressiveness*). Beide Faktoren werden von Reynolds von 0 bis 100 Prozent bewertet. Dabei legt risk fest, ob potentiell unsichere Ereignisse abgearbeitet werden, während aggressiveness regelt, ob die Ergebnisse unsicherer Berechnungen an andere LPs weitergereicht werden. Konservative Verfahren arbeiten somit ohne Risiko und Aggressivität (beide bei 0 Prozent). Der Time Warp hingegen belegt auf beiden Skalen die Maximalwerte.

Dickens und Reynolds schlagen als Erweiterung des konservativen SRADS-Protokoll ein Verfahren mit lokalen Rollbacks vor (SRADS/LR) [DIR90a], das zwar risikobehaftet aber frei von aggressivem Verhalten ist. Lokal werden wie bei Time Warp alle Ereignisse sofort verarbeitet. Allerdings werden Ereignisse, die für andere LPs generiert werden, erst dann verschickt, wenn ausreichende Garantien vorliegen. Im Fehlerfall erfolgt dann lediglich ein lokaler Rollback ohne Notwendigkeit, Antinachrichten zu verschicken. Ähnliche Ansätze verfolgen *Local-time-warp* [RAT93a] sowie *Breathing-time-buckets* [STE91a] und *Breathing-time-warp* [STE93a].

Local-time-warp arbeitet dabei ähnlich wie SRADS/LR, jedoch werden zusätzlich alle lokalen LPs als Cluster betrachtet und die clusterlokalen Ereignisse ausgeliefert. Breathing-time-buckets arbeitet in Phasen mit globaler Synchronisation aller Prozesse und erlaubt die Ausführung aller lokaler Ereignisse bis zum Zeitstempel des ersten innerhalb der aktuellen Phase neu generierten Ereignisses.

Diese beiden ersten Verfahren arbeiten gänzlich ohne Aggressivität, letzteres steuert über zwei fest vorgegebene Schwellwerte Aggressivität und Risiko. Überschreitet die LVT im Breathing-time-warp die erste Stelle, wird die Aggressivität abgeschaltet. Bei Erreichen des zweiten Grenzwerts blockiert die Simulation gänzlich. In die Klasse der hybriden Verfahren fallen z.B. auch die fensterbasierten Erweiterungen des Time Warp, bei denen Risiko und Agressivität begrenzt werden [SBW88a].

Auf Basis hybrider Verfahren ist auch eine applikationsspezifische Kombination von im Extremfall rein konservativen und rein optimistischen LPs innerhalb eines Modells möglich [ARS91a]. Ist z.B. von einem LP bekannt, daß viele kausal voneinander abhängige oder zustandsverändernde Ereignisse auftreten können, so liegt eine konservative Strategie zur Bearbeitung nahe. Wird andererseits weitgehend lesend auf die Zustandsgrößen zugegriffen, so bietet sich eine optimistische Variante an.

Mehl entwickelte von konservativen Protokollen ausgehend das Verfahren der spekulativen Simulation [MEH91a], bei dem im Fall einer drohenden Blockade auf einer Kopie des Zustandes spekulativ weitersimuliert wird. Tritt kein Fehler auf, der die Spekulation widerlegt, so konnte die sonst in der Blockade verbrachte Zeit für sinnvolle Arbeit verwendet werden. Die Änderungen des Zustands werden in die Originale übernommen. Ebenso werden Ereignisse für andere LPs erst dann in die zugehörigen Originaldaten übertragen und verschickt. Im Fehlerfall wird die Kopie verworfen und mit der Simulation auf dem Ausgangszustand fortgefahren.

2.6.3 Adaptive Methoden

Einen noch stärkeren Applikationsbezug bzw. die Berücksichtigung dynamischer Information und Daten, deren Werteverlauf erst zur Simulationszeit vorliegt, weisen adaptive Verfahren auf. Die Steuerung zur Festlegung des Optimismus der Simulationsverfahren ist hier von verschiedenen dynamischen Messungen zur Laufzeit abhängig. Wichtige Größen in diesem Zusammenhang sind die Anzahl und Tiefe der Rollbacks, die durchschnittliche Erhöhung der Simulationszeit, der Beitrag von Rechenzeit zur Erhöhung der GVT (effektive Prozessorauslastung) [REJ90a], die Anzahl aufgetretener Deadlocks, das Verhältnis von Null-, Anti- oder sonstigen Kontrollnachrichten zur Anzahl der Ereignisnachrichten [FUJ88b]. Als davon abhängige Regelgrößen können die Breite des Fensters bei Window-Verfahren [RAT93a], Checkpointabstände [LIL89a, LPL93a, PAW93a], Speicherbedarf [DAF97a] oder die Häufigkeit des Propagierens von Garantien modifiziert und den Bedürfnissen zur Laufzeit angepaßt werden. Bei Über- bzw. Unterschreiten gewisser Grenzwerte kann somit der Optimismus reduziert oder erhöht und somit der vom Simulationsverfahren generierte Aufwand gesteuert werden. Allerdings ist generell zu bemerken, daß insbesondere bei Verwendung mehrerer Steuergrößen eine globale Optimierung vor allem in einer verteilten Umgebung nicht trivial ist.

2.6.4 Weiterführende Literatur

Es existiert eine Reihe von Übersichtsartikeln, die die existierenden Simulationsalgorithmen diskutieren. Im Bereich der konservativen Verfahren beschreiben Misra [MIS86a] und Reynolds [REY88a] Basisverfahren und Erweiterungen. Righter und Walrand [RIW89a] und Fujimoto [FUJ90a] greifen diese Thematik ebenfalls auf, stellen ihr jedoch auch ausführlich die optimistischen Ansätze gegenüber. Weitere Arbeiten werden von Fujimoto und Nicol in [FUN92a, NIF94a] und [FUJ95a] beschrieben. Zu Space-time existiert neben dem Originalartikel von Chandy und Sherman im wesentlichen nur ein weiteres Paper von Bagrodia, Chandy und Liao [BCL91a]. In [MEH94a] schließlich wird eine Übersicht der hybriden Verfahren gegeben.

2.7 Konzeptionelle Probleme bei der parallelen Simulation

2.7.1 Terminierungserkennung

Das Ende einer Simulation kann im sequentiellen Fall trivialerweise durch Überprüfen einer Abbruchbedingung (meist durch Festlegung eines Maximalwerts der Simulationszeit) festgestellt werden, da der Simulator ein globales Wissen über die Zeit und den Zustand aller simulierten Objekte hat. Sobald keine Ereignisse mit kleinerem Simulationszeitpunkt als der Endzeitpunkt mehr vorhanden sind, ist der Simulationslauf beendet. Im verteilten Fall stellen sich bei den unterschiedlichen Paralleliserungsansätzen verschiedene Probleme.

Bei der reinen zeitgesteuerten Simulation (ohne Ermittlung des Zeitpunkts des nächsten Ereignisses) überprüft jeder Simulator nur seine lokale Ereignisliste. Ohne besondere Kooperation ist also nicht feststellbar, ob spätere Ereignisse noch generiert werden. Deshalb läuft die Simulation ggf. etliche Zeitschritte unnötig weiter, bis die Abbruchbedingung erfüllt ist ohne daß jedoch ausführbare Ereignisse existieren. Immerhin verfügen in zeitgesteuerten Verfahren alle Simulatoren über ein gemeinsame quasi-globale Uhr, so daß durch Einführen eines virtuellen Zeitstempels *unendlich* (∞) ein Simulator in den ereignisorientierten Erweiterungen bei der Ermittlung des lokal kleinsten Ereignisses diesen Wert liefern kann, sofern er keine Ereignisse mehr zu bearbeiten hat. Existieren im gesamten Modell keine Ereignisse mehr, so liegt das Minimum aller Zeitstempel entweder bei einem größeren Wert als der Stoppzeitpunkt, oder es ist ebenfalls ∞ und übersteigt somit jeden

Terminierungszeitpunkt. Die Erkennung der Terminierung wird somit in den Grundalgorithmus eingebettet, ohne daß komplexere Synchronisationen zwischen einzelnen Simulatoren nötig sind.

Etwas komplizierter wird das Problem für die reine ereignisgesteuerte Simulation. Hier besitzen die einzelnen LPs in der Regel voneinander abweichende lokale Simulationszeiten – repräsentiert durch die Local-virtual-time (LVT). In konservativen Verfahren steht die LVT für den Zeitstempel des zuletzt abgearbeiteten Ereignisses. Optimistische Verfahren verwenden den Zeitstempel des kleinsten lokal noch nicht ausgeführten Ereignisses als LVT. Außerdem existieren keine streng getrennten Synchronisations- und Berechnungsphasen wie bei der Zeitsteuerung. Deshalb muß über einen unterliegenden Terminierungerkennungsmechanismus neben der Überprüfung der zeitlichen Abbruchbedingung anhand der Ereignislisten auch sichergestellt werden, daß keine Nachrichten mit kleineren Zeitstempeln im System mehr unterwegs, d.h. gesendet aber noch nicht empfangen, sind. Die Erkennung kann mit einem der bereits beschriebenen Deadlockerkennungsalgorithmen kombiniert werden, die ja zur Auflösung von Deadlocks den Zeitstempel des global kleinsten Ereignisses liefern. Erreicht dieser Wert einen größeren Zeitstempel als den Terminierungszeitpunkt, so kann die Simulation beendet werden.

Wird lediglich ein Deadlock-avoidance-Verfahren eingesetzt, so zeigt sich bei kleinem Lookahead ein ähnlich schlechtes Verhalten wie beim zeitgesteuerten Ansatz, da die Zeit bis zum Terminierungszeitpunkt in durch den Lookahead festgelegten kleinen Inkrementen durch Nullnachrichten fortgeschaltet werden muß. Alternativ kann ein LP auf allen Ausgangskanälen auch eine Kontrollnachricht mit Zeitstempel ∞ verschicken, sofern er aufgrund seines lokalen Wissens die Terminierung feststellen kann. Existiert gar kein Lookahead, so ist insbesondere in zyklischen Modellen ein zusätzlicher Terminierungserkennungsalgorithmus unabdingbar, um das Simulationsende (und ggf. Deadlocks) festzustellen.

In optimistischen Verfahren kann analog zur Deadlockerkennung die GVT zur Überprüfung der Terminierung eingesetzt werden.

2.7.2 Gleichzeitige Ereignisse und Kausalität

Die Behandlung des Zeitmodells und der zulässigen Zeitstempel für Simulationsalgorithmen wird in vielen Arbeiten oft vernachlässigt. In diesem Abschnitt werden zunächst die Arten der Verzögerungsmodellierung, d.h. die Regeln für die Bestimmung von Zeitstempeln für Folgeereignisse bei der Ereignisgenerierung beschrieben. Danach sollen die Probleme kausal abhängiger Ereignisse mit gleichem Zeitstempel erörtert werden.

2.7.2.1 Zeitstempelgenerierung

Durch ein Delaymodell wird festgelegt, nach welchen Regeln ein Ereignis die Zeitstempel für von ihm selbst erzeugte Ereignisse ermittelt. Die Art und Weise, in der die zeitliche Auflösung in einem Modell nachgebildet wird, ist essentiell für den verfügbaren Parallelismus und die Kausalitätsbetrachtungen unterschiedlicher Ereignisse.

Die einfachste Zeitmodellierung erfolgt gemäß dem *Unit-delay*-Verfahren. Alle Ereignisse werden auf eine Einheitszeitskala abgebildet. Eine explizite Modellierung von Simulationszeit liegt nicht vor. Ein neu generiertes Ereignis wird stets mit einem um 1 erhöhten Zeitstempel, d.h. im folgenden Simulationsschritt, ausgeführt. Alle Ereignisse, die zum selben Zeitpunkt abgearbeitet werden, müssen somit als kausal unabhängig betrachtet werden. Sie wurden alle zum vorhergehenden Zeitpunkt generiert. Diese Kette läßt sich bis zu den initial vorhandenen Ereignissen zurückverfolgen. Es ist somit lediglich eine Reihenfolgebetrachtung aber keine echte Nachbildung eines realen Zeitmodells möglich.

Bei *Fixed-delay* wird jedem Objekt ein während der gesamten Simulation konstanter Verzögerungswert zur Erzeugung der Simulationszeitstempel nachfolgender Ereignisse aus der aktuellen Simulationszeit zugeordnet. Der konstante Wert 0 ist möglich. In einer Simulation mit Ereignisgenerierung nach diesem Verfahren werden Ereignisse mit demselben Ausführungszeitstempel in der Regel von Ereignissen zu unterschiedlichen Zeitpunkten generiert. Eine Aussage über die zu erwartende Anzahl der Ereignisse für den Zeitpunkt t_i ist nur noch nach statistischen Verfahren approximierbar.

Bei *Variable-delay* schließlich wird die Verzögerungszeit dynamisch für jedes Ereignis zur Simulationslaufzeit berechnet. Die Abschätzung des Parallelismus wird dadurch noch unschärfer. Eine typische Anwendungsklasse hierfür sind Bedienzeiten in Warteschlangennetzen. Die Verzögerung "0 Zeiteinheiten" kann ebenfalls auftreten. Die Möglichkeit, Ereignisse mit dem identischen Zeitstempel des Erzeugers zu generieren, bereitet allerdings bei der Sicherstellung der kausalen Abhängigkeiten ebenso wie bei Fixed-delay einige Probleme, auf deren Behandlung später eingegangen wird.

Coehlo [COE89a] beschreibt die verschiedenen Modelierungsmöglichkeiten für die Schaltkreissimulation anhand der Hardwarebeschreibungssprache VHDL [VHD87a]. Die Verzögerung bei Variable-delay hängt hier vor allem von der Anzahl der treibenden und getriebenen Objekte eines Schaltelements ab.

2.7.2.2 Behandlung gleichzeitiger Ereignisse

Bei ereignisgesteuerten Verfahren werden gelegentlich folgende Voraussetzungen bezüglich der Zeitstempelwerte gemacht [JEF85a]:

- Ereignisse eines Simulators haben auch bei verschiedenen Quellen paarweise verschiedene Zeitstempel bezogen auf einen lokalen Simulator,
- Ereignisse generieren nur neue Ereignisse mit höheren Zeitstempeln für sich selbst und andere LPs.

Grundgedanke bei diesen Einschränkungen ist die Garantie der Reproduzierbarkeit von Simulationsläufen. Wenn mehrere Ereignisse mit gleichem Zeitstempel auftreten, können bei beliebiger Abarbeitungsreihenfolge unterschiedliche Ergebnisse produziert werden, sofern zwischen den Ereignissen kausale Abhängigkeiten bestehen.

Die (z.B. von Jefferson) gemachten Annahmen sind jedoch sehr restriktiv und können bei vielen Modellierungen nur durch zusätzliche künstliche Modifikationen des Modells erreicht werden. In Warteschlangenmodellen z.B. macht es durchaus Sinn, daß zwei Ereignisse aus verschiedenen Quellen gleichzeitig bei einer Queue eintreffen dürfen. Eine weitere Einsatzmöglichkeit ist z.B. die Modellierung eines idealen Servers, der seine Dienste in Nullzeit erbringt und die bearbeiteten Jobs mit dem ursprünglichen Ankunftszeitpunkt an ein nachgeschaltetes Verzögerungsglied weiterreicht.

In diesen Fällen muß, sofern Determinismus benötigt wird, durch Erweiterung des Zeitstempels um weitere Komponenten eine verbindliche Reihenfolge von Ereignissen definiert werden. So verwendet VHDL, das auch zur Beschreibung der Modelle für unsere prototypische Implementierung eines parallelen Simulators dient, einen zweistufigen Zeitstempel, der ein sogenanntes Delta-delay neben der eigentlichen Simulationszeit einführt, um Ereignisse mit gleichem Primärzeitstempel in eine feste Reihenfolge zu bringen.

Erzeugt die Bearbeitung eines Ereignisses ein neues Ereignis zum selben Zeitpunkt, so wird ähnlich wie bei Unit-delay der Zyklus um eins erhöht. Bei einem neuen Ereignis mit größerem

Zeitstempel ist der Deltawert initial Null. Alle Ereignisse innerhalb desselben Deltazyklus wurden somit von tatsächlich vorangegangenen Ereignissen generiert. Sie sind somit kausal unabhängig und können in beliebiger Reihenfolge abgearbeitet werden. Bei direkter Verknüpfung mehrerer Signale (z.B. Wired-or) wird eine explizite vom Modellierer zu definierende Arbitration-Funktion benötigt, die eine deterministische Entscheidung über die Reihenfolge bei gleichzeitigen Ereignissen aus verschiedenen Quellen trifft.

Mehl beschreibt in [MEH91c] einen allgemeinen Ansatz zum Erweitern von Zeitstempeln, um Determinismus zu erreichen. Jefferson schlägt eine Erweiterung entsprechend Lamports logischer Zeit vor [JEF85a]. Bagrodia u.a. [BCL91a] gestatten die Existenz von Ereignissen, die keinen Beitrag zum Fortschalten der Simulationszeit leisten und neue Ereignisse mit demselben Zeitstempel generieren können, unter der Prämisse, das es nur endlich lange Folgen solcher Ereignisse gibt. Als Beispiel wird eine begrenzte Anzahl von Ausführungen eines Jobs in Warteschlangen genannt. Somit wird sichergestellt, daß die Simulationszeit letztendlich doch fortschreitet. Bryant verwendet eine Entscheiderfunktion zur eindeutigen Reihenfolgefestlegung, wenn auf verschiedenen Eingangskanälen Nachrichten mit gleichem Zeitstempel eingehen [BRY79a]. Im Time-warp-operation-system (TWOS)[4] [JBW87a] wurde ebenfalls eine Komponente zur Sicherstellung der deterministischen Ausführung von Programmläufen integriert [RWH90a], indem der Programmierer eine Entscheidungsfunktion für alle Nachrichten mit gleichem Zeitstempel in den Simulator einklinken kann.

2.7.3 Fehlender Lookahead

Wie bereits in den vorhergehenden Abschnitten kurz erwähnt, bereitet fehlender Lookahead insbesondere bei konservativen Verfahren erhebliche Probleme, die hier noch einmal zusammengefaßt werden sollen.

- Für Deadlock-avoidance-Protokolle ist ein von Null verschiedener Lookahead unbedingt notwendig [CHM81a]. Andernfalls drohen bei Existenz von Zyklen Verklemmungen.
- Misra beschreibt in [MIS86a] Deadlock-detection und -recovery und postuliert einen Lookahead, der von Null verschieden ist, als wünschenswerte, wenn auch nicht zwingende Eigenschaft.

Bei optimistischen Verfahren werden keine Annahmen über Lookahead gemacht. Varghese, Chamberlain und Weihl [VCW94a] beschreiben jedoch interessante Analogien zwischen Lookahead in konservativen Verfahren und effektiven Verfahren zur GVT-Berechnung in optimistischen Systemen in Abhängigkeit vom verfügbaren Lookahead.

2.8 Hardwareunterstützung

Einen vollkommen unterschiedlichen Ansatz, der in den frühen achtziger Jahren entwickelt wurde, stellt die Realisierung spezieller Simulationsalgorithmen in Hardware oder die Unterstützung von Simulationen durch spezielle Hardwarekomponenten dar. Daß dieser Ansatz durchaus Berechtigung hat, zeigen auch heutige Entwicklungen in anderen Bereichen, z.B. Grafikkarten mit eingebauter 3D-Funktionalität, Decoder für Audio- und Videodatenströme (MPEG) oder spezielle Chips zur

[4] TWOS realisiert ein speziell auf die Belange der optimistischen Simulation zugeschnittenes Betriebssystem. Einen weiteren Ansatz, Simulationsumgebungen in ein eigenes Betriebssystem zu betten, stellt Mimdix von Madisetti, Hardaker und Fujimoto dar [MHF92a].

Datenver- und -entschlüsselung (Clipper). Hochberger [HOC97b] entwickelte an der TU Darmstadt eine PC-Karte mit einem programmierbaren Zellularautomaten, der prinzipiell für die zeitgesteuerte Simulation oder Simulation nach dem Unit-delay-Verfahren von Objekten mit maximal 4 Eingängen und 4 Ausgängen verwendet werden könnte (CEPRA-1X). Als Beispielapplikation sind hier Gefechtsfeldsimulationen denkbar, die ihre Ein- und Ausgabedaten nur mit den unmittelbaren Nachbarn austauschen müssen.

Für eine seinerzeit typische RISC-Workstation (5-8 MIPS) mit Einschubkarten zur hardwaremäßigen Simulationsunterstützung wurden 1986 von Smith Ereignisraten zwischen 50000 und 300000 Ereignissen pro Sekunde erzielt [SMI86a]. Er schlägt außerdem die Verwendung von Parallelrechnern mit Simulationshardwarekomponenten zur weiteren Geschwindigkeitssteigerung auf 2 Millionen Ereignisse in einer Sekunde mit 32 Prozessoren vor.

Im Bereich der Simulation wurden jedoch nicht nur einzelne Zusatzkomponenten für konventionelle Rechner, sondern komplette Rechnerarchitekturen entwickelt, die spezielle Simulationsanwendungen realisierten. Hier sollen exemplarisch zwei Rechner kurz beschrieben werden.

IBM stellte 1982 mit der Yorktown Simulation Engine (YSE) einen Schaltkreissimulator vor, der auf Gatterebene zur Beschleunigung konventioneller softwaremäßiger Simulationen führte. Mit bis zu 256 über einen Crossbar-Switch gekoppelten Logikprozessoren arbeitete die YSE um bis zu 1000-mal schneller als ein herkömmliches Simulationsprogramm auf einem IBM Mainframe-Rechner (IBM 370). Es wurden dabei über 3 Millionen Gatterauswertungen pro Sekunde ausgeführt. Die Speicherkapazität umfaßte bis zu 2 Millionen Gatter.

Die Ansteuerung der Prozessoren erfolgte dabei in SIMD-Manier über einen zentralen Hostrechner. Die Gatter werden beginnend bei den primären Eingängen rangartig in Abhängigkeit von der minimalen Entfernung zu einem solchen Eingang sortiert. Innerhalb eines Simulationszykluses werden die Gatter beginnend bei den Primäreingängen rangweise ausgewertet und die neu produzierten Ausgangssignale zu den Eingängen des folgenden Rangs propagiert. Es sind jedoch nur zyklenfreie kombinatorische Netzwerke für die Simulation vorgesehen, da jedes Gatter pro Zyklus genau einmal ausgewertet wird. Aufgrund des zeitgesteuerten Ansatzes werden hier ggf. viele unnötige Berechnungen ausgeführt, da sich die meisten Werte nicht in jedem Zyklus ändern [BAS88a]. Dadurch relativieren sich auch die angegebenen Leistungsdaten wiederum. Interessant wäre hier zu wissen, wieviele effektiv nötige Auswertungen im Mittel durchgeführt werden.

Eine zusätzliche Beschränkung, die die YSE dem Anwender auferlegt, ist die ausschließliche Zulassung von Gattern mit vier Eingängen und einem Ausgang. Die Modellierung hat diesem (vermutlich aus Vereinfachungsgründen für die Realisierung der Logikprozessoren) gewählten Ansatz Folge zu tragen und muß zusätzliche Eingangsbelegungen für "Blindeingänge" identifizieren.

MuSiC (Munich-simulation-computer) [HAF85a] wurde von 1985 – 1990 an der Universität Passau entwickelt und führt eine parallele Simulation von Schaltkreisen auf einem Datenflußrechner mit 256 Prozessoren durch. Die Schaltung wird dazu in einen zyklenfreien Kontrollflußgraphen aufgebrochen, der ebenfalls rangweise durch den Rechner abgearbeitet wird. Es werden dabei ca. 10^{10} Ereignisse pro Sekunde ausgeführt. Weitere Ansätze für hardwaremäßige Beschleunigungsversuche durch Nutzung von Spezialrechnern finden sich in [FWW84a] und [SMI86a].

Neben kompletten Rechnern wurden weiterhin einzelne Komponenten mit dedizierten Aufgaben zur Simulationsunterstützung realisiert. Am Jet Propulsion Laboratory wurde der Rollback-Chip [FTG88a] zur Unterstützung des TWOS entwickelt. Dabei handelt es sich um ein besonderes Speichermodul, das neben der eigentlichen Adressierung über die Speicheradresse die Simulationszeit als weiteren Adreßparameter benötigt. Für alle vom Modellierer besonders gekennzeichneten Variablen werden die neuen Werte für jeden mit einem noch nicht verwendeten Zeitstempel durchgeführten Schreibvorgang in einer neuen physikalischen Speicherstelle abgelegt. Bei einer Leseoperation wird

dann anhand des Zeitstempels der Wert der bezüglich der Simulationszeit letzten unmittelbar davor stattgefundenen Schreiboperation geliefert. Das Rücksetzen der Zustandsvariablen nach einem Rollback erfolgt durch einfaches Setzen der lokalen Uhr auf den früheren Zeitpunkt. Danach sind beim Speicherzugriff die alten Werte automatisch wieder verfügbar. Für komplexere Datenstrukturen, insbesondere die Input- und Output-Queue, wird keine implizite Unterstützung angeboten.

Zur Ausführung unterliegender Algorithmen wie beispielsweise GVT, Deadlockerkennung oder Terminierung schlagen Pancerella und Reynolds [PAR93a] einfache über einen Bus gekoppelte Logikprozessoren vor, mit denen sehr schnell die benötigten Werte berechnet werden können. Dadurch wird der Overhead der softwarebasierten verteilten Algorithmen vermieden. In [SRR95a] wird ein solcher Algorithmus zur schnellen Berechnung der GVT beschrieben. Er erlaubt aufgrund der unterstützenden Hardware, die GVT sehr schnell zu berechnen, wodurch die einzelnen Prozessoren den Simulationsfortschritt besser beurteilen können. Durch ein Bremsen überoptimistischer LPs in darauf aufbauenden Simulatoren werden sich kaskadierende Längen von Rollbacks und ein instabiles Time Warp-System vermieden, so daß die falschen Berechnungen sich nicht gegenseitig hochschaukeln können. Es bleibt somit genug Zeit, um Rollbacks entsprechend Lubachewskys Vorschlag zu behandeln [LSW89a]. An der Universität Brest wurde ein ähnlicher Ansatz auf Basis eines unterhalb der eigentlichen Parallelrechnerschicht liegenden Transputernetzwerks verfolgt [BBC94b].

Obwohl für die meisten dieser hardwareorientierten Projekte vielversprechende Beschleunigung berichtet werden, hängt ihnen das Makel der kostenintensiven Speziallösungen an, die nicht wie reine Softwarelösungen auf Basis einer portablen Programmierplattform universell auf beliebigen Rechnern einsetzbar sind. Im Hinblick auf die Verwendbarkeit für beliebige Modelle verlangen die Hardwareunterstützungen ein weitgehendes Verständnis für deren Funktionsweise beim Entwickler eines Simulationsmodells. Um die Vorteile wirklich ausnutzen zu können, ist eine entsprechende Anpassung der modellbezogenen Algorithmen und Zugriffsverfahren auf die Hardwarekomponenten nötig. Die Simulationsmodelle können somit auch nur noch unter erhöhtem Anpassungsaufwand für andere Architekturen wiederverwendet werden. Vermutlich aus den genannten Gründen konnten sich diese Ansätze auch nicht in größerem Stil durchsetzen.

Kapitel 3

Kriterien zur Beurteilung von Beschleunigungspotential

Im vorherigen Kapitel wurden verschiedene Verfahren zum Beschleunigen von Simulationsläufen beschrieben, ohne auf wichtige Faktoren, die bei diesen Verfahren Aufwand generieren, einzugehen. Weiterhin wurde bislang keine konkrete Definition des Begriffs *Beschleunigung* und die Festlegung einer Bewertungsfunktion dafür vorgenommen. Beides ist jedoch wesentlich für die Beurteilung der Leistungsfähigkeit sequentieller und paralleler Simulationsverfahren. Die Vorgabe entsprechender Definitionen sowie ein Überblick über verschiedene eingesetzte Metriken zur Bewertung der Simulationsverfahren ist Inhalt dieses Kapitels.

3.1 Kosten der sequentiellen Simulation

Ein Simulationslauf besteht im sequentiellen Fall aus einigen wenigen Bearbeitungsabschnitten, die sich je nach Art des Simulatortyps unterscheiden. Existiert zu einer Menge verschiedener Modelle ein generischer Simulator, dessen Programmcode universell zur Simulation aller Modelle eingesetzt werden kann, so muß jede Modellbeschreibung zu Simulationsbeginn vom Simulator eingelesen, analysiert und in entsprechende interne Objektrepräsentationen konvertiert werden. Erst danach startet die eigentliche Simulation des Modells. Die Kosten für das Einlesen eines Modells fallen somit bei jedem Simulationslauf an.

Alternativ dazu existieren Verfahren, die ein Modell vorkompilieren, mit einem generischen Simulatorteil zusammenbinden und somit für jedes Modell ein eigenes Simulationsprogramm erzeugen. Über parametrisierte Variablen läßt sich innerhalb gewisser Grenzen ein modifiziertes Verhalten eines solchen Simulators auch ohne erneute Übersetzung realisieren. Die Kosten der Kompilierung fallen somit nur einmal für verschiedene Simulationsläufe desselben Modells an. Allerdings muß das Modell bei jeder Änderung der Beschreibung ebenfalls wieder kompiliert werden, was im Entwicklungszyklus zu erhöhtem Zeitbedarf führt.

Der für unsere Untersuchungen zugrundeliegende sequentielle VHDL-Simulator VSIM [LEV93a] verwendet beide Verfahren. Für einfachere mit VHDL-Strukturbeschreibungen erstellte Modelle genügt ein generischer Simulator. Wird die Schaltung jedoch zusätzlich über ihr dynamisches Zeitverhalten ("behavioral view") [VHD87a] beschrieben, muß ein dedizierter Simulator generiert werden, der vom VHDL-Kompiler erzeugten modellspezifischen C-Code einbindet. Anschließend kann die eigentliche Simulation durchgeführt werden. Die Aufbereitung und Ausgabe der Ergebnisse erfolgt dabei, sofern das möglich ist, entweder bereits zur Laufzeit oder aber in einer Postprocessing-

Phase im Anschluß. Es kann also davon ausgegangen werden, daß die wesentliche Zeit für die eigentliche Simulationsaufgabe verwendet wird und lediglich kurze Intervalle zu Beginn und Ende für Zusatzaufgaben verwendet werden.

Auf die Optimierungsproblematik sequentieller Simulatoren soll hier nur am Rand eingegangen werden. Fujimoto weist darauf hin, daß durch geeignete Modellierung der Ereignistypen erhebliche Beschleunigungen bereits in sequentiellen Simulationen erreicht werden können [FUJ88b]. Als Beispiel führt Fujimoto das Ersetzen von Start-Stopp-Paaren, die den Beginn und das Ende von Simulationszeit benötigenden Aktionen modellieren, durch die Kombination beider Aktionen innerhalb eines einzigen Startevents an. Dieses plant dann selbst direkt das nächste Startereignis ein, z.B. bei Realisierung einer FIFO-Strategie für eine Warteschlange. Dadurch kann die Modellierung von Aktivitäten, die eigentlich durch zwei Ereignisse, Start und Stopp, repräsentiert werden, aufgrund geschickter Abbildung und Vorausberechnung von Bedienzeiten lediglich mit einem einzigen Startevent, also global auf die gesamte Simulation übertragen mit der Hälfte der Ereignisse auskommen (Abb. 3.1).

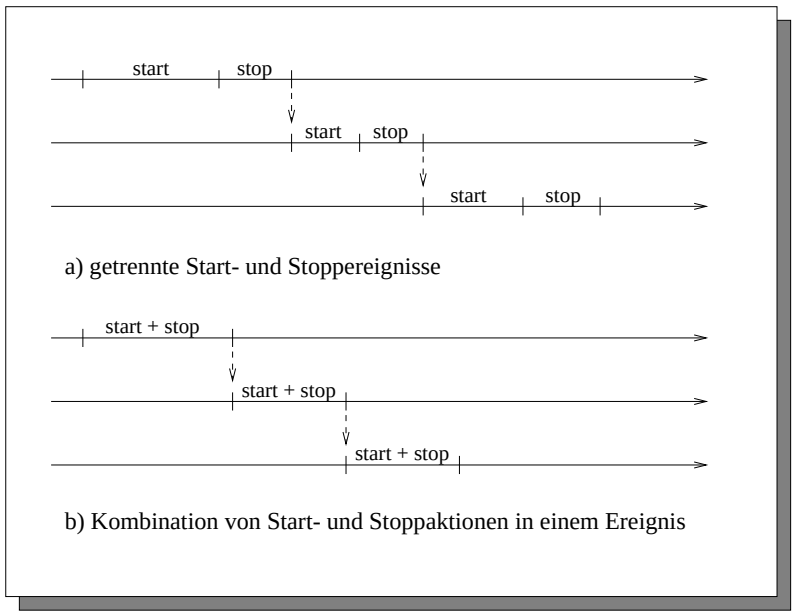


Abbildung 3.1: Optimierung der Ereignisabhängigkeiten

Außerdem gilt es natürlich bereits in sequentiellen Simulatoren möglichst effiziente Datenstrukturen z.B. zur Realisierung der Ereignisliste zu verwenden. Andernfalls können hohe Beschleunigungen durch die Parallelisierung beobachtet werden, die schlichtweg auf der durch die Aufteilung der Ereignisliste bedingten Reduzierung der Listenlänge beruhen und prinzipiell nichts mit der eigentlichen Parallelisierung zu tun haben. Diese Effekte wurden bei Ferscha und Richter [FER96a] ebenso wie beim Entwurf unseres eigenen parallelen Simulators auf der Basis von VSIM [LEV93a] beobachtet. Nachdem in VSIM eine effizientere Verwaltungsstruktur eingefügt wurde (die natürlich auch in der verteilten Version Verwendung fand), reduzierte sich die sequentielle Laufzeit drastisch und der zunächst beobachtete superlineare Speedup von DVSIM war verschwunden. Ferscha und Richter berichten dagegen von Speedups in der Größenordnung 300 mit 32 Prozessoren. Der Einfluß der Listenverwaltung wird zwar genannt, jedoch wird scheinbar keine Optimierung des sequentiellen Algorithmus durchgeführt. Dieses Beispiel zeigt, daß man derart überschwengliche Ergebnisse kritisch zu überprüfen hat.

Als Maß für die Simulationsdauer dient im konventionellen sequentiellen Fall somit die Länge eines Programmlaufs (gemessen in physikalischer Zeit) einschließlich Vor- und Nachbearbeitung. In der Literatur wird jedoch nicht immer angegeben, ob diese beiden zusätzlichen Phasen in den Meßergebnissen, die zu Beschleunigungsvergleichen herangezogen werden, berücksichtigt sind oder nicht. Der Vergleich von Simulationszeitdauern wird ebenfalls in parallelen Verfahren eingesetzt, wo der Anteil zusätzlicher Bearbeitungsschritte allerdings weitaus höher ist. Die dabei zu berücksichtigenden Einflußgrößen werden in Abschnitt 3.2 diskutiert.

Ein weiteres Maß für den Vergleich der Leistung verschiedener sequentieller und paralleler Simulationsalgorithmen ist die Anzahl der bearbeiteten Ereignisse. Da es sich bei vergleichenden Betrachtungen um Lösungen desselben Simulationsproblems handelt, ist die durchschnittliche Bearbeitungsdauer eines Ereignisses (Granularität) prinzipiell vergleichbar und dieses Parallelitätsmaß für den Vergleich der reinen Simulationsphase daher gerechtfertigt.

Eine von der Granularität losgelöste Definition der Leistung allein über die reine Anzahl der Ereignisse macht allerdings für verschiedene Implementierungen einer Simulation keinen Sinn, da durchaus Effizienzunterschiede bei verschiedenen Realisierungen (z.B. aufgrund der von Fujimoto beschriebenen optimierten Ereignisstruktur) auftreten können. Läßt sich beispielsweise ein unoptimierter Simulationslauf mit 20 Ereignissen der Granularität 100 ms ausführen, während eine optimierte Variante mit 10 Ereignissen, von denen jedes 150 ms benötigt, auskommt, so erhält man, skaliert auf die Zeit, im ersten Fall 10 Ereignisse pro Sekunde und im zweiten 6,6 Ereignisse pro Sekunde. Dennoch ist die Simulationszeit im zweiten Ansatz kürzer. Für die Untersuchungen in dieser Arbeit wird deshalb vorwiegend die Simulationsdauer als Maßstab verwendet.

Insbesondere bei der Verteilung der Simulationskomponenten kann eine Erweiterung des Simulationsmodells um zusätzliche Ereignistypen gegenüber einer sequentiellen Bearbeitungsstrategie nötig werden. Beim Vergleich zwischen paralleler und sequentieller Version könnte die Ereignismetrik mitunter auch zu irreführenden Schlußfolgerungen verleiten. Ein direkter Vergleich ist hierüber also nur im Fall eines identischen Ereignismodells möglich.

3.2 Kosten der parallelen Simulation

Parallele Simulationsverfahren stellen gegenüber der Simulation auf nur einem Rechner weitaus höhere Anforderungen an die Vor- bzw. Nachbearbeitungsschritte des eigentlichen Simulationslaufs.

Vor den reinen Simulationsberechnungen muß eine Unterteilung des Modells in einzelne (in der Regel disjunkte) Submodelle erfolgen. Die Bedeutung der Partitionierung für die Qualität der erreichbaren Beschleunigung ist in der Fachwelt insbesondere für große Modelle unbestritten [BBC94a, NIF94a]. Im Bereich der Schaltkreissimulation ist eine Partitionierung von Hand bereits für kleinste Schaltungen recht mühsam und in der Regel daher nur durch Automatisierung handhabbar. Insbesondere kommen bei der Aufteilung eines Simulationsmodells teilweise ähnliche Algorithmen zum Tragen wie bei der Plazierung von Schaltungselementen auf einem Chip [FIM82b, KEL70a, SES85a]. Die Kosten für die Berechnung der jeweiligen Aufteilung sind gerechtfertigterweise in die Gesamtkosten der Simulation mit einzubeziehen. In den Veröffentlichungen ist dieser Faktor bis auf wenige Ausnahmen [MSD89a, MAL93a, KAA92a] bisher leider vernachlässigt worden, was ungerechtfertigterweise höhere Beschleunigungswerte für die parallelen Simulationsläufe liefert. Eine optimale Partitionierung wird in den meisten Fällen nicht erreichbar sein, da die Kosten für ihre Berechnung in der Regel extrem hoch sind. Merkmale für Partitionskosten sind bei der parallelen Simulation Aufwand für die Interprozeßkommunikation oder die Submodellgröße in Abhängigkeit vom erwarteten zu leistenden Rechenaufwand. Wichtige Partitionierungsverfahren für den Bereich der parallelen Simulation, die auch in der vorliegenden Arbeit

Eingang fanden, werden in Kapitel 5 beschrieben. Die zu erwartende Qualität der Lösungen sowie die Kosten der Berechnung werden ebenfalls abgeschätzt (Kapitel 8.4). Der zusätzliche Partitionierungsaufwand kann sich teilweise amortisieren, wenn Simulationsläufe mit derselben Aufteilung aber unterschiedlichen Eingaben (Stimuli) mehrfach ausgeführt werden.

Der zweite Kostenfaktor, der bei im Rahmen von verteilten Simulationsläufen noch vor der eigentlichen Simulationsphase anfällt, besteht in der Verteilung der Submodellkomponenten auf die unterschiedlichen Prozessoren. Entsprechend der Partitionierungsinformation erhalten die einzelnen Rechnerknoten die für sie relevanten Modelldaten zugewiesen. Das kann zum einen über einen zentralen Kontrollprozeß, der auch das Einlesen und Parsen des Modells übernimmt, geschehen. Andererseits kann das Einlesen und Verteilen auch parallelisiert werden, indem jeder Prozeß selbst die für ihn anhand der Partitionierungsdaten erkennbaren relevanten Daten einliest. Beschleunigung ist dann erreichbar, wenn z.B. der Partitionierungsalgorithmus die Modelldaten jeder Partition in einer eigenen Datei speichert und das echt parallele Lesen auf dem Dateisystem von der Rechnerhardware unterstützt wird. Ein solcher verteilter Ansatz ist jedoch wohl nur für sehr große Modelle nötig und setzt eine physikalische Verteilung der Teilmodellbeschreibungen auf verschiedenen Speichermedien voraus, die von den einzelnen Prozessoren separat angesprochen werden können (paralleles Dateisystem, Workstation-Cluster mit lokalen Platten, usw.).

Im Anschluß an den parallelen Simulationslauf treten zwei weitere, im sequentiellen Fall ebenfalls nicht existente Berechnungsteile auf. Zunächst muß die Terminierung der einzelnen Teilsimulatoren festgestellt werden. Dazu werden die bereits im letzten Kapitel erwähnten Terminierungserkennungsalgorithmen eingesetzt. Anschließend sammelt in der Regel ein zentraler Prozessor die Ergebnisse der Simulation ein und gibt sie wie bei der sequentiellen Simulation in aufbereiteter Form aus oder speichert die Resultate.

Eine parallele Anwendung kann deshalb nur dann als beschleunigbar gelten, wenn sie unter Berücksichtigung aller dieser zusätzlichen Aspekte eine kürzere Laufzeit als ihr sequentielles Pendant hat. Ein fairer Vergleich zwischen beliebigen parallelen oder sequentiellen Verfahren legt daher die Summe der benötigten Zeiten für folgende Einzelberechnungen zugrunde:

- Partitionierung t_{part},
- Einlesen des Modells t_{input},
- Verteilung des Modells $t_{distrib}$,
- reine Simulation t_s,
- Terminierungserkennung t_{term},
- Einsammeln der Ergebnisse $t_{collect}$ und
- Aufbereitung und Ausgabe der Ergebnisse t_{output}.

Im sequentiellen Fall treten einige Aufgaben prinzipiell nicht auf oder nehmen nahezu keine Zeit in Anspruch. Die Evaluierungsdauer des Simulationsmodells t_s und die Zeit zur Terminierungserkennung t_{term} sind meßtechnisch oft ebenfalls schwer zu trennen. Die Summe der beiden Werte wird deshalb mit t_{sim} benannt.

Die Bewertung existierender Verfahren soll in der vorliegenden Arbeit unter diesem Gesichtspunkt der umfassenden Berücksichtigung aller anfallenden Kosten erfolgen. Für eine gegebene Hardware, auf der die Simulation abläuft, hängt das Beschleunigungspotential somit von den realen Laufzeiten der zu vergleichenden Programme unter den o.g. Kriterien ab.

Die Beschleunigung wird als Verhältnis der sequentiellen Rechenzeit t zur parallelen Gesamtlaufzeit T betrachtet[1]. Die Summe der Einzelwerte erhalten im sequentiellen Fall keine Indizes und bei der parallelen Simulation mit n Prozessoren den Index n.[2]

Als Grundformel zur Berechnung der erreichten Beschleunigung mit n Prozessoren, dem *Speedup*, gilt also:

$$Speedup_n = \frac{t}{T_n}.$$

T_n setzt sich dabei aus der Summe aller Einzelwerte $T_{part,n}$, T_{input}[3], $T_{distrib,n}$, $T_{sim,n}$, $T_{collect,n}$ und $T_{output,n}$ zusammen. Es muß hier nochmals betont werden, daß diese Summenbildung notwendig ist, um die tatsächlich für einen Anwender beobachtbaren Beschleunigungsfaktoren zu erfassen, die die parallele Simulation über den gesamten Bearbeitungsprozeß bietet. Viele Veröffentlichungen berücksichtigen lediglich das Verhältnis von t_{sim} oder sogar $T_{sim,1}$ zu $T_{sim,n}$ und verfälschen die Ergebnisse zugunsten der parallelen Verfahren.

3.3 Zeitliches Beschleunigungspotential der Simulationsläufe

Unabhängig von den hier genannten zusätzlichen Randbedingungen bei der parallelen Simulation ist grundsätzlich auch die Frage interessant, ob bereits in einem Simulationslauf genügend Potential vorhanden ist, um prinzipiell eine Beschleunigung durch Parallelisierung zu erreichen. Wäre schon diese Voraussetzung nicht gegeben, so lohnt sich eine parallele Simulation nicht. Auf die diskrete ereignisgesteuerte Simulation übertragen bedeutet dies: Existieren genügend kausal unabhängige Ereignisse, die zeitgleich von unterschiedlichen Prozessoren bearbeitet werden können, so daß sich der Aufwand der Parallelisierung überhaupt rechnet, oder ist in der Regel ein Großteil der Prozessoren nicht ausgelastet, weil sie z.B. in konservativen Verfahren auf Garantien durch andere LPs warten? Der Speedup der reinen Simulation beträgt in diesem Fall:

$$Speedup_{sim,n} = \frac{t_{sim}}{T_{sim,n}}$$

und entspricht dem vielfach in der Literatur verwendeten Wert. Liegt der so ermittelte Faktor bereits nahe bei "1", ist das ein Anzeichen dafür, daß die parallele Simulation sich für einen Anwender nicht rentieren wird, da die Zusatzkosten ebenfalls noch hinzuzurechnen sind. Der Umfang dieser Kosten für die sequentielle und parallele Simulation wird in den Kapiteln 7.2 und 8.4 anhand konkreter Benchmarks beschrieben.

3.3.1 Parallelismus im Simulationsmodell

Wong u.a. [WFC86a] untersuchten das Parallelisierungspotential für zeitgesteuerte Simulation anhand verschiedener Schaltkreise auf der Gatter- und der Transistorebene elektrischer Schaltungen (gate- und switch-level). Sie fanden bei Modellen mit 650 bis 7950 Transistoren eine Aktivitätsrate der Schaltungselemente zwischen 0,08 und 3,1 Prozent für jeden synchronen Zeitschritt, in dem

[1] Im folgenden werden Variablen, die die Laufzeit im parallelen Fall beschreiben, mit dem Großbuchstaben T und entsprechendem Index gekennzeichnet, während die Werte bei sequentieller Bearbeitung mit t und zugehörigem Index markiert sind.

[2] Die Anzahl der verwendeten Prozessoren kann dem Index bei den Einzelwerten ggf. durch ein mit Komma abgetrenntes n nachgestellt werden. So stellt $T_{part,4}$ die Kosten zur Berechnung der Partitionen für 4 Rechnerknoten dar.

[3] Das Einlesen des Modells erfolgt zentral und ist unabhängig von der Prozessoranzahl.

wenigstens ein Ereignis eingeplant war. Das bedeutet, daß, wenn überhaupt etwas zu tun war, im Mittel der angegebene Anteil aller Schaltelemente ein Ereignis ausführen konnte.

In 86 Prozent der Zeitschritte waren jedoch keine Ereignisse vorhanden. Diese Zeitpunkte müssen jedoch bei der reinen zeitgesteuerten Simulation ebenfalls abgearbeitet werden, was unnötigen Zusatzaufwand bereitet und ein typisches Kennzeichen dafür ist, daß eine ereignisgesteuerte Kontrolle einer synchronen Zeitfortschaltung vermutlich überlegen ist.

Die relativ niedrigen Aktivitätsraten ergeben jedoch skaliert auf die gesamte Schaltungsgröße einen potentiellen Parallelitätsgrad von 0,5 bis 20,5 Ereignissen beim kleinsten und 6,3 bis 246,5 Ereignissen beim größten Schaltkreis. Diese theoretischen Mittelwerte versprechen scheinbar gute Parallelisierbarkeit. Bei den untersuchten Schaltkreisen waren tatsächlich jedoch nur maximal zwischen 7 und 35 gleichzeitige Ereignisse pro Simulationsschritt vorhanden (und eben nicht 20,5 bis 246,5).

Bailey und Snyder [BAS88a] untersuchten sechs Schaltkreise der Größe 200 bis 27000 Transistoren und fanden ähnliche Aktivitätsraten (0,04 – 2,9 Prozent). Allerdings ergaben weitere Analysen, daß im Schnitt nur etwa 5 Ereignisse pro Zeitschlitz vorhanden waren, wobei allerdings auch leere Zeitschritte bei der Mittelwertbildung berücksichtigt wurden. Eine wesentliche Ursache wird dem sequentiellen Verhalten der untersuchten Schaltkreise (Multiplexer, RISC-Multiprozessor, digitaler Filter) zugeschrieben, die in 45 Prozent aller Zeitschritte nur genau ein Ereignis zur Simulation besaßen.

Da beide Untersuchungen jedoch mit zeitgesteuerter Simulation erfolgten, wurde ggf. auch zusätzliches Parallelisierungspotential, das die ereignisgesteuerten Varianten durch ihre Optimierung bei der Fortschaltung der Simulationszeit und der virtuell gleichzeitigen Bearbeitung von Ereignissen mit unterschiedlichen Zeitstempeln bieten, "unterschlagen".

3.3.2 Kausal begründeter Parallelismus (Kritischer Pfad)

Untersuchungen der Beschleunigungsmöglichkeiten nach dem asynchronen ereignisgesteuerten Ansatz wurden von Berry und Jefferson mittels *Kritische-Pfad-Analyse* durchgeführt (CPA = critical path analysis) [BEJ85a]. Entsprechend der kausalen Abarbeitungsreihenfolge und Dauer der Ereignisbearbeitung werden die frühsten Realzeitpunkte berechnet, zu denen ein Ereignis (unabhängig von seinem Simulationszeitstempel) bearbeitet werden kann. Dies ist entweder sein Generierungszeitpunkt, wenn der Prozessor, auf dem die Bearbeitung erfolgt, keine Ereignisse mit kleinerem Simulationszeitstempel mehr zuvor auszuführen hat, oder der frühste Zeitpunkt, zu dem der entsprechende Prozessor alle diese Ereignisse verarbeitet hat.

Die Kommunikationszeiten t_{msg} für den Versand der Ereignisnachrichten zwischen den einzelnen Prozessoren werden als konstant angenommen und sind im einfachsten Fall Null. Als Generierungszeitpunkt eines Ereignisses wird der reale Endzeitpunkt der Bearbeitung des generierenden Ereignisses betrachtet. Trägt man die Bearbeitungszeiten der Ereignisroutinen auf Prozessorenachsen ab, so werden lokal bearbeitete, aufeinanderfolgende Ereignisse lückenlos aneinandergereiht. Ereignisse, die für andere LPs eingeplant werden, erhalten im Diagramm einen senkrechten (oder um die Nachrichtenlaufzeit verzögernden) Verbindungspfeil, falls der Ziel-LP zu diesem Zeitpunkt keine anderen Ereignisse mehr zu bearbeiten hat. Andernfalls wird die kausale Abhängigkeit der Ereignisse durch einen Pfeil zum Endzeitpunkt aller zuvor auf dem Ziel-LP zu simulierenden Events repräsentiert. Die Mindestlaufzeit eines verteilten Simulators entspricht somit dem kritischen Pfad durch dieses Diagramm. In Abhängigkeit von der Anzahl der Prozessoren und der Partitionierung ergeben sich natürlich unterschiedliche Ergebnisse.

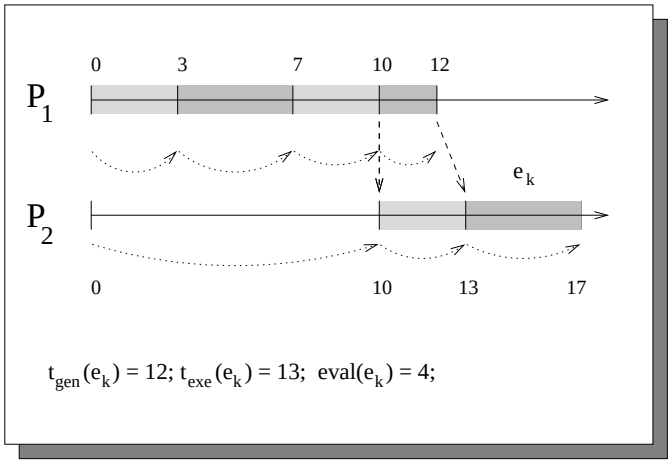


Abbildung 3.2: Komponenten bei der Berechnung des kritischen Pfads

Die Berechnung des kritischen Pfads kann durch Erweiterung der Datenstrukturen des sequentiellen Simulators um einige Variablen nahtlos in diesen integriert werden. Dazu werden für jeden Prozessor der Realzeitpunkt der letzten Aktivität und für jedes Ereignis sein realer Generierungs- und Ausführungszeitpunkt gespeichert. Die Berechnung des kritischen Pfads erfolgt mit nachstehendem informellen Algorithmus.

- Der Wert $t_{last}(P_j)$ beschreibt den Zeitpunkt, zu dem Prozessor P_j die letzte Ereignisbearbeitung abgeschlossen hat. Ist e_k das nächste auszuführende Ereignis, so gibt $t_{last/e_k}(P_j)$ den neu berechneten Wert von $t_{last}(P_j)$ nach der Ausführung dieses Ereignisses an. Dauert die Berechnung der Ereignisroutine von e_k die Zeit $eval(e_k)$, so ist:

  $$t_{last/e_k}(P_j) = t_{last}(P_j) + eval(e_k).$$

  Nach Abarbeitung von e_k übernimmt $t_{last}(P_j)$ den neu ermittelten Wert $t_{last/e_k}(P_j)$.

- Für die Berechnung des Generierungszeitpunkts eines Ereignisses e_2 für Prozessor P_i, das von e_1 auf Prozessor P_j erzeugt wurde (wobei i = j sein kann), gilt somit folgende Beziehung:

  $$t_{gen}(e_2) = t_{last/e_1}(P_j).$$

  Der Generierungszeitpunkt ergibt sich also aus dem Endzeitpunkt der Bearbeitung des generierenden Ereignisses.

- Ist $E = \{e_1, \ldots, e_n\}$ die Menge der vor Ereignis e_k abzuarbeitenden Ereignisse auf dem ausführenden Prozessor P_j, so wird e_k nicht vor deren Bearbeitung, das heißt frühestens zum Realzeitpunkt $t_{ready}(e_k)$ ausgeführt:

  $$t_{ready}(e_k) = \max_{e_i \in E}\{t_{last/e_i}(P_j)\}.$$

  t_{ready} muß nicht explizit berechnet werden, da sich der Wert automatisch aus der Abarbeitungsreihenfolge der vorhergehenden Ereignisse ergibt und dem Wert von t_{last} nach Simulation aller Elemente von E entspricht.

- Für den frühsten Ausführungszeitpunkt t_{exe} eines beliebigen Ereignisses gilt folgende untere Schranke:

$$t_{exe}(e_2) \geq \max\{t_{ready}(e_2),\ t_{gen}(e_2) + t_{msg}\}.$$

  Das Gleichheitszeichen gilt, wenn e_2 das Ereignis mit dem kleinsten Simulationszeitstempel auf dem Ziel-LP P_i ist und dieser bei Ankunft der Ereignisnachricht mit e_2 kein Ereignis bearbeitet.

Wird nun jeder LP auf einem eigenen Prozessor ausgeführt, so kann jedes Ereignis unmittelbar ausgeführt werden, nachdem es verfügbar und alle anderen lokalen Ereignisse bearbeitet wurden. Mit einer unbeschränkten Prozessoranzahl und somit einer Partitionierung P_∞, bei der jeder LP auf einen eigenen Rechnerknoten abgebildet wird, erhält man also den maximal erreichbaren Speedup einer asynchronen parallelen Simulation durch Vergleich der Länge des kritischen Pfades t_{cpa,P_∞} mit der Summe der Bearbeitungsdauer aller Ereignisroutinen, die der Länge der sequentiellen Bearbeitung t_{sim} entspricht:

$$Speedup_{opt,P_\infty} = \frac{t_{sim}}{t_{cpa,P_\infty}}.$$

Werden mehrere LPs einem Prozessor zugeteilt, verlängert sich aufgrund der gemeinsam genutzten Ressourcen in der Regel auch die Länge des kritischen Pfads, da ggf. zwischenzeitlich noch Ereignisse der anderen LPs bearbeitet und somit die Verarbeitung eigener Events verzögert wird. Alle LPs eines Prozessors P_i arbeiten dann auf einem gemeinsamen Wert $t_{last}(P_i)$. Die Beschleunigung ergibt sich somit unter einer gegebenen Partitionierung P_n zu:

$$Speedup_{opt,P_n} = \frac{t_{sim}}{t_{cpa,P_n}}.$$

Der kritische Pfad mit $t_{msg} = 0$ ist für konservative Simulationsverfahren eine untere Schranke der möglichen Laufzeit. Als realistischer Overhead sind die Nachrichtenlaufzeiten oder der Synchronisationsaufwand jedoch durch einen von Null verschiedenen Wert für t_{msg} zu berücksichtigen. Für optimistische Verfahren wurden bei Verwendung von Lazy-cancellation jedoch auch Szenarien beschrieben, in denen der kritische Pfad unterboten werden kann [JER91a, GUN94a], da Ereignisse bereits verfrüht ausgeführt werden können und durch Rollbacks nicht wieder zurückgesetzt werden, weil sie später wieder identisch generiert werden. Dadurch kann der Zielsimulator bereits früher weiterarbeiten als ein konservatives Verfahren und unterschreitet die Länge des kritischen Pfads.

Lin beschreibt für verschiedene Schedulingstrategien bei Plazierung mehrerer LPs auf einem Prozessor die Vorgehensweise zur Berechnung des kritischen Pfads [LIN92a]. Es wurden jedoch nur Untersuchungen mit kleinen Warteschlangennetzen unter Verwendung verschiedener Parametrisierungen der Job- und Prozessoranzahl sowie der Topologie durchgeführt. Bei entsprechend hoher Population des Warteschlangennetzes mit einer ausreichenden Anzahl von Jobs wurden Speedups des kritischen Pfads von 20 – 25 mit 32 Prozessoren beobachtet.

3.3.3 Oracle-log

Das eben beschriebene Verfahren berücksichtigt allerdings in keiner Weise die *realen* Kosten der Kommunikation, den Aufwand für Deadlockerkennung und Blockadezeiten. Diese Faktoren werden entweder komplett vernachlässigt oder als konstant angenommen. Swope und Fujimoto schlagen deshalb für konservative Synchronisationsverfahren eine Methode vor, die auf Basis eines "perfect

knowledge" stets in der Art und Weise eines allwissenden Orakels angeben kann, ob das nächste zu simulierende Ereignis sicher ist, oder ob noch Ereignisse von anderen Simulatoren ausstehen [SWF87a]. Auf diese Weise vermeidet das "Oracle-log"-Verfahren Blockaden, die sich im nachhinein als unnötig herausstellen, da die Simulation nur dann blockiert, wenn wirklich auf Ereignisse anderer LPs gewartet werden muß.

Da der Algorithmus als echter paralleler Simulator läuft, werden auch die realen Nachrichtenlaufzeiten berücksichtigt. Aufgrund des vollständigen Wissens, das aus einem Protokoll (Log) der Ereignisreihenfolge eines vorherigen (sequentiellen oder parallelen) Simulationslaufs herrührt, wird ebenfalls entschieden, ob auf Nachrichten gewartet werden muß oder nicht. Der Gedanke, daß auf einen Deadlockerkennungs- oder -vermeidungsalgorithmus verzichtet werden kann, trifft leider nicht zu. Warten zwei Simulatoren auf Nachrichten mit gleichem Zeitstempel vom jeweils anderen LP, so blockieren sie dennoch.

Durch Vergleich der Simulationsdauer des quasioptimalen verteilten konservativen Oracle-log mit der Laufzeit eines konventionellen konservativen Verfahrens läßt sich ein Maß für den Overhead eines konkret implementierten parallelen Algorithmus angeben. Von praktischer Bedeutung in Produktionsumgebungen ist das angegebene Verfahren jedoch nicht, da stets ein zusätzlicher (und in der Regel langsamerer) Simulationslauf für die Protokollierung der Ereignisreihenfolge nötig ist. Richter beschreibt ausführlich die Leistungsverluste, die beim Übergang von optimalen hin zu realistischen konservativen Verfahren entstehen können [RIC95a]. Er berichtet von Verlusten in Größenordnungen zwischen 60 und 85 Prozent der im Idealfall erreichbaren Beschleunigung. Allerdings werden nur relativ kleine Warteschlangenmodelle mit 420 bis 2800 Objekten, deren Partitionierung von Hand erfolgte, betrachtet. Es stellt sich jedoch die Frage, ob die Aussagen auf größere Modelle, die ggf. ein wesentlich höheres Potential gleichzeitiger Ereignisse bieten, direkt übertragen werden können.

Dasselbe Verfahren wird erstaunlicherweise auch in identischer Weise nochmals von Jha und Bagrodia unter dem Titel *Ideal-simulation-protocol* [JHB96a] im Rahmen der Maisie-Simulationsumgebung [BJW94a] eingeführt. Die Beobachtung für ein geschlossenes Warteschlangennetz ist hier, daß ein reales konservatives Synchronisationsprotokoll auf Basis von Nullnachrichten auch bei Variation verschiedener Parameter (Lookahead, Topologie, Jobs und Granularität) lediglich zwischen 40 und 60 Prozent der mit dem idealen Protokolls erreichbaren Beschleunigung erzielt. Bei diesen Untersuchungen werden ebenfalls lediglich 16 LPs zugrundegelegt.

3.3.4 Weitere Bewertungskriterien

3.3.4.1 Laufzeiten des parallelen Simulators

Neben dem von uns verwendeten Beschleunigungsmaß für reine Simulationslaufzeiten, das dem Verhältnis der Laufzeit eines optimierten sequentiellen Simulators zu der eines verteilten Simulators auf n Prozessoren in Abhängigkeit von einer gegebenen Partitionierung entspricht, wurden von verschiedenen Forschern insbesondere bis Anfang der 90er Jahre noch einige andere Definitionen von Speedup angegeben.

Reed, Malony und McCredie [RMM88a] verwenden als Referenzwert statt des sequentiellen Simulators die Laufzeit $T_{sim,1}$ des parallelen Simulators mit einer einzigen Partition unter der Begründung, daß dieser Wert eine obere Schranke für die Beschleunigung liefert. Je nach Qualität dieser verteilten Version auf einem einzigen Prozessor kann diese Schranke allerdings um einiges höher liegen als real erreichbare Werte. In unserer bereits stark optimierten Version des parallelen Simulators DVSIM läuft der verteilte Simulator auf einem einzigen Knoten zwischen 10 und 30 Prozent langsamer als die rein sequentielle Version. Ähnliche Werte berichten auch Bagrodia und

Jha [BAJ96a]. Allerdings mußte hierzu der parallele Simulator bereits sehr stark optimiert werden. Bauer und Sporrer [BAS93a] zeigen, daß mit dieser Bewertung die erreichbaren Speedup-Werte mitunter "verdoppelt" werden können, wenn die parallele Referenzimplementierung relativ langsam zur sequentiellen Version ist.

Mittlerweile hat sich jedoch der Vergleich zur sequentiellen Variante weitgehend durchgesetzt. Wo dennoch der parallele Simulator auf einem Knoten als Basis dient, wird als Äquivalent das Verhältnis der Leistung des sequentiellen Simulators zur parallelen Version auf einem Knoten angegeben, so daß sich die Ergebnisse nachträglich noch vergleichen lassen [FUJ88b, LUB89a, RIW89a, BSK92a, JHB96a].

Wagner, Lazowska und Bershad [WLB88a] legen ebenso wie Bagrodia, Chandy und Liao [BCL91a] die verteilte Variante zugrunde. Über deren Qualität wird keine Aussage getroffen. In [JHB96a] wird jedoch auch von Bagrodia eine optimierte sequentielle Version sowie ein idealer auf dem Ansatz des kritischen Pfads beruhender Simulator als Referenz verwendet.

3.3.4.2 Effizienz

Als weitere Bewertungsgröße nennt Lubachewsky die Effizienz, die als Maß der Auslastung der genutzten Ressourcen die erreichte Beschleunigung dem optimalen Speedup n bei Verwendung von n Prozessoren gegenüberstellt [LUB89a]:

$$eff_n = \frac{Speedup_n}{n}.$$

[RIW89a] nennen die Effizienz ebenfalls als Maß des Erreichbaren. Durch die Abstraktion von der konkreten Prozessoranzahl lassen sich Simulationsergebnisse verschiedener Forschergruppen auch bei Angabe von Ergebnissen, die auf unterschiedlichen Rechnerknotenanzahlen erzielt wurden, vergleichen. Speedup-Werte von beispielsweise 2,8 auf 5 Prozessoren und 9,8 auf 16 Knoten liefern als Effizienzmaß 56 bzw. 61 Prozent und und sind damit leichter einzuordnen.

3.3.4.3 Ereignisraten

Wie bereits diskutiert, werden neben Simulationszeiten auch die Anzahl der bearbeiteten Ereignisse als Kriterium verwendet. Bei konservativen Verfahren werden dazu alle Ereignisse gezählt. Matsumoto und Taki erreichten mit diesem Bewertungsmaßstab Speedups von 50 auf 64 Prozessoren unter Time Warp [MAT92a]. Dabei wurden allerdings durch Rollback widerrufene Ereignisse anscheinend mitgezählt. Der korrigierte Wert ohne solche Events beträgt 43 und nach einer weiteren Korrektur der Zählweise nur noch 35. Insgesamt scheinen die Ergebnisse eher etwas konfus als wirklich fundiert. Zudem existiert keine Angabe, welche Messungen als Basis der Vergleiche dienten (sequentielle Version oder verteilter Simulator mit einer Partition).

3.3.4.4 Sonstige Maßstäbe

[CAP94a] definiert losgelöst von bestimmten Simulationsanwendungen unterschiedliche Bewertungsmaßstäbe für die Parallelisierbarkeit und allgemeine Bewertung paralleler Algorithmen auf unterschiedlichen Rechnerarchitekturen. Insbesondere werden die Begriffe der idealen Laufzeit, der kommunikationsfreien Laufzeit, der Effizienz (vgl. [LUB89a]), des Scaleup (vgl. Isoefficiency bei [KGR94a]) und des sequentiellen Anteils an der Gesamtlaufzeit betrachtet. Außerdem wird ähnlich wie bei [LIN92a] die Leistung unterschiedlicher Maschinen innerhalb eines heterogenen Rechnerverbunds berücksichtigt.

3.3.5 Leistungsbewertung und Abgrenzung paralleler Verfahren untereinander

3.3.5.1 Konservative Methoden

Zur Bewertung implementierter Synchronisationsverfahren und zur Abgrenzung ihrer Leistungsfähigkeit gegenüber Varianten oder anderen Verfahren können ebenfalls unterschiedliche Größen betrachtet werden. Reed, Malony und McCredie stellen als Maß für die Effizienz eines Verfahrens die Anzahl der Deadlocks und der Kontrollnachrichten der Gesamtanzahl aller Nachrichten gegenüber [RMM88a]. Sie berechnen somit die Anzahl der Deadlocks pro Nachricht bzw. den prozentualen Anteil der Nullnachrichten am Gesamtaufkommen für Kommunikation. Beide Werte gilt es, für die effiziente Realisierung eines konservativen Algorithmus zu minimieren.

Fujimoto verwendet die Inversen der beiden eben genannten Quotienten als Rate der reinen Ereignisnachrichten pro Deadlock bzw. pro Nullnachricht [FUJ88a], die dann zu maximieren sind. Er zeigte auch, daß echte Parallelverarbeitung von Ereignissen erst ab einem gewissen Kommunikationsaufkommen von Ereignisnachrichten erreicht werden kann, da dann alle Prozessoren stets genügend Ereignisse in ihren Event-Listen verfügbar haben. Somit entfallen Blockaden leerlaufender Prozesse und der Parallelisierungsaufwand amortisiert sich. Steigen die Nachrichtenraten, so erhöhen sich bei Fujimotos Untersuchungen auch die Speedup-Werte der betrachteten Warteschlangennetze.

3.3.5.2 Meßgrößen optimistischer Verfahren

Ähnlich den protokollorientierten Größen Nullnachrichten und Deadlocks bei konservativen Verfahren stehen bei optimistischen Methoden die den Zusatzaufwand verursachenden Rollbacks und Sicherungspunkte im Blickpunkt der Leistungsbetrachtungen.

Vielfach wird die Anzahl der Rollbacks in Kombination mit der Rollbacktiefe als Gütekriterium herangezogen [APW93a] (für Workstation-Cluster), [CGU94a] (in Shared-memory-Umgebung). Die Rollbacktiefe wird dabei als mittlere Anzahl der zurückgesetzten Ereignisse angegeben oder über die zurückgesetzte Simulationszeit ausgedrückt. Während das erste der beiden Maße unabhängig von der konkreten Simulation und ihrer Zeitmodellierung ist, empfiehlt sich bei Verwendung der Simulationszeit eine Skalierung z.B. anhand der Gesamtdauer der Simulation.

Ein ebenfalls geeignetes Maß für den Aufwand, den Rollbacks verursachen, ist die CPU-Rechenzeit für die Ausführung der Rücksetzaktionen im Vergleich zur Gesamtrechenzeit der parallelen Simulation. Weiterhin lassen sich Rollbackkosten nach primären Rollbacks, die aufgrund verspäteter Ereignisnachrichten ausgelöst werden, und sekundären Rollbacks, die durch vorherige Rücksetzungen anderer LPs mittels Antinachrichten beim Empfänger verursacht werden, unterteilen.

Bei der Bewertung des Aufwands für Sicherungspunkte beurteilt man in der Regel den Rechenzeitaufwand für ihrer Erstellung und Verwaltung sowie in einigen Fällen auch den Speicherbedarf.

3.4 Untersuchte Modellklassen

Die Modelle, die bislang auf Eignung für die Parallelisierung untersucht wurden, lassen sich grob in folgende Klassen einteilen: Kommunikationsnetzwerke, Warteschlangennetze, Gefechtsfeldsimulationen und ähnliche Modelle, die auf topographischen Daten aufsetzen, sowie Simulation von Schaltkreismodellen. Einige Ergebnisse zur Untersuchung dieser Modellklassen sollen hier kurz skizziert werden.

- Aufgrund der fundierten Theorie, anhand derer sich Untersuchungen auch teilweise analytisch überprüfen lassen, stellen Warteschlangennetze die wohl beliebtesten Forschungsobjekte dar. Die Größe der Modelle ist dabei meist auf weniger als 1000 LPs, die jeweils eine Warteschlange repräsentieren, beschränkt.

  – Fujimoto betrachtet in [FUJ88a] 4 x 4 und 8 x 8 torusförmig angeordnete Warteschlangen mit initial 4 Jobs pro Queue. Die Granularität der Ereignisroutinen beträgt 1 ms und wird im Programm über aktive Warteschleifen realisiert. Als Rechnerplattform stand ein Butterfly BBN Shared-memory Parallelrechner zur Verfügung. Auf ihm wurde ein konservatives Verfahren mit Lookahead realisiert und Untersuchungen für Deadlock-avoidance durch Nullnachrichten und Deadlock-detection und -recovery untersucht. Die Deadlockerkennung wurde dabei durch den gemeinsamen Speicher wesentlich erleichtert. Tabelle 3.1 zeigt die erreichten Speedup-Werte der Implementierung.

Algorithmus / Prozessoren	Deadlock-avoidance	Deadlock-detection
4	3.5	2.5
8	6.5	4
16	11	5.8
Referenz: sequentieller Simulator		

Tabelle 3.1: Speedup-Ergebnisse bei Fujimoto

    Insbesondere wurde der Einfluß der Granularität und des Nachrichtenaufkommens auf die Beschleunigung untersucht. Ab einer gewissen Sättigung des Systems mit Nachrichten wird eine Verbesserung der Beschleunigung beobachtet, die auf eine Erhöhung der verfügbaren Parallelität zurückgeführt wird. Die Prozessoren verfügen dann stets über genügend Ereignisse, so daß die Ereignislisten nicht leer werden und somit effektiv Ereignisse gleichzeitig bearbeitet werden.

    Außerdem spielt der verfügbare Lookahead eine nicht unwichtige Rolle. Es werden unterschiedliche Ankunftsraten mit verschiedenen Lookaheads untersucht. Bei *Lookahead-ratio* ist lediglich das Verhältnis zwischen den mittleren Abständen zweier Ereignisse und dem Lookahead relevant und nicht allein der absolute Lookahead-Wert. Je näher das Verhältnis an "1" liegt, um so weniger zusätzlicher Aufwand ist zur Freigabe sicherer Ereignisse nötig.

    Bei gutem Lookahead-Verhältnis (nahe bei 1) ist die Deadlockvermeidungsstrategie der Deadlockerkennungsvariante überlegen (um den Faktor 2 schneller). Wichtig ist stets, die "Nachrichtenlawine" (Message-avalanche, [FUJ88a]) auszulösen, damit genug parallele Ereignisse vorhanden sind. Das Mapping der LPs erfolgte clusterweise nach zusammenhängenden Komponenten.

  – Ein ebenfalls bei Forschern beliebtes Modell unter den Warteschlangennetzen ist der sog. Central-Server-Ansatz, in dem eine zentrale Queue (z.B. CPU) Jobs bearbeitet und diese anschließend nach einer definierbaren Wahrscheinlichkeitsverteilung an Servicestationen (z.B. I/O-Geräte) verteilt. Nach der Bearbeitung an diesen Stationen kehren die Jobs wieder zur Central Queue zurück. Danach wiederholt sich der Zyklus.

    Fujimoto betrachtet in [FUJ88b] ein Central-Server-Modell mit 5 Knoten sowie ein Routing-Modell in einem Hypercube-Netzwerk. Er führt die in Abschnitt 3.1 genannte

Optimierung der Eventmodellierung bei FCFS-Queues durch Verwenden zusätzlichen Applikationswissens ein. Bereits bei der Ankunft eines Jobs kann das Ende der Bedienzeit und somit die Ankunft beim nächsten LP vorausberechnet werden. Events können somit früher generiert und bearbeitet werden. Implizit erhöht sich dadurch der Lookahead. Mit dieser Optimierung erreicht Deadlock-detection einen Speedup von 3 für das Central-Server-Modell mit 5 Knoten. Beim Routing-Modell erzielen die beiden konservativen Verfahren Speedups von 5 bei 8 Prozessoren, wenn 64 LPs modelliert wurden. Bei Verwendung eines Hypercubes mit 16 LPs ergaben sich schlechtere Werte. [RMM88a] erzielten wesentlich schlechtere Ergebnisse für das Central-Server-Modell. Die Autoren verwendeten aber auch keine Optimierung der Ereignisstruktur wie Fujimoto.

  - Bagrodia [BCL91a] erreichte mit seinem Space-time-System für die Simulation von Warteschlangennetz mit Zyklen einen Speedup von 14 mit 16 Prozessoren bei alleiniger Verwendung räumlicher Parallelität. Das Verfahren wird zwar als asynchron bezeichnet, synchronisiert sich aber dennoch gelegentlich mit allen anderen Simulationsprozessen (SP) über Barrieren. Ein gewisser Optimismus ist vorhanden, der über Checkpointing (im einfachsten Fall eine Kopie des Initialzustands) abgesichert wird. Für ein Feed-forward-Warteschlangennetz ohne Zyklen wurde bei extrem geringem Kommunikationsaufkommen und unter Einsatz der zeitlichen Parallelisierung mittels Relaxationsverfahren zur Konvergenzsicherung ein Speedup von 54 auf 64 Prozessoren erreicht. Durch die geringe Interaktionsrate werden entsprechend wenig Fehler gemacht und die initiale Schätzung des Zustands ist oft korrekt [BCL91a].
  - Anhand einer Reihe weiterer Warteschlangenmodelle untersuchten insbesondere Lin, Lubachewsky und Lazowska [LIL89a, LIL89b, LIL89c, LIL89d, LIL90c, LLB89a, LLB89b, LIL90b, LSW89a] prinzipielle Eigenschaften von Modellen mit zwei (oder nur wenigen) Warteschlangen und stellten daraus eine Reihe von Hypothesen zur Leistungsfähigkeit paralleler Simulationen auf, die sich jedoch wegen vieler Einschränkungen zur Vereinfachung der gefundenen Formeln schwer auf die Praxis übertragen lassen.

- Kommunikationsnetze klassischer Art, aber auch Funknetze, bedürfen bei der Planung ebenfalls sorgfältiger Untersuchungen und wurden für parallele Simulationsstudien verwendet. Das klassische Problem der Simulation von Kommunikationsnetzen veranlaßte Bryant 1977, die auch nach ihm mit benannte CMB-Methode der parallelen ereignisgesteuerten Simulation zu entwickeln [BRY77a]. [FUJ89b] beschreibt die parallele Simulation von Mobilfunk-Netzen. Im TeleSim-Projekt soll durch Einsatz der optimistischen PDES die Simulation von ATM-Netzen beschleunigt werden [WUZ98a].

- Viele Forschungsprojekte im Bereich der parallelen Simulation wurden (und werden im Rahmen der High Level Architecture (HLA) auch jetzt wieder) aus US-Regierungsmitteln finanziert. Klassischerweise ist das amerikanische Verteidigungsministerium an Simulationen militärischer Aktionen interessiert. Eines der größten Projekte wurde im Rahmen des Timewarp-operating-system (TWOS) am Jet Propulsion Laboratory realisiert [WHF89a]. Gefechtsfeldsimulationen spielen dabei die wichtigste Rolle (ein Standardmodell zur Simulation zweier gegeneinander kämpfender Armeen hat die Bezeichnung STB88).

  - In [WIJ89a] erfolgt ein Vergleich zwischen zeit- und ereignisgesteuerten Verfahren. Es werden 225 Objekte als Zellen auf einem Gitterraster simuliert. Die Ankunft von Einheiten (Units) aus Nachbarzellen wird über Ereignisse modelliert.

Es wurden folgende Speedups beobachtet: Die ereignisgesteuerte Simulation lieferte Beschleunigungen von 5 bei 32 Prozessoren auf einem Caltech JPL Mark III Hypercube mit TWOS. Die parallelen zeitgesteuerten Versionen waren stets langsamer als die optimierten sequentiellen Version. Im Gegensatz zu Aussagen von Briner [BRI90a] ist die sequentielle ereignisgesteuerte Simulation bereits ca. 35 Prozent schneller als die reine kompilierte zeitgesteuerte Variante. Briner stellt bei einer durch Überspringen von Leerzeiten optimierten zeitgesteuerten Version eine ca. 10-fach schnellere Bearbeitung gegenüber der diskreten ereignisgesteuerten Variante fest.

– Einen analogen Ansatz, der auf der Simulation kleiner geographischer Regionen mit Grenzen zu den Nachbargebieten beruht, stellt Shark's World dar. Bagrodia erreichte mit einer zeitgesteuerten Variante seines Space-time-basierten Simulationssystems Maisie Beschleunigungen von 4.8 auf 32 Prozessoren [BCL91a]. Als Referenzwert dient jedoch die Einprozessorversion des verteilten Simulators.

- Als Modelle zur Schaltkreissimulation werden oft proprietäre Entwicklungen, die an den Standorten der einzelnen Forschergruppen verfügbar waren, verwendet. Dadurch ist eine Vergleichbarkeit verschiedener Ansätze schwierig. Allerdings erfreuen sich in den vergangenen Jahren die Schaltungen der ISCAS'85- [BRF85a] und ISCAS'89-Benchmark-Suite [BBK89a] zunehmender Beliebtheit. Leider ist bei diesen Schaltungen die konkrete Funktionalität nicht bekannt. Sie waren ursprünglich zur automatischen Testmustergenerierung und Fehlersimulation zusammengestellt worden. Die meisten Untersuchungen verwenden deshalb Zufallsmuster als Stimuli. Die beiden Suites enthalten zum einen zustandslose kombinatorische Modelle (ISCAS'85) sowie auch zustandsbehaftete sequentielle Schaltungen (ISCAS'89). Die Größe der Schaltungen liegt für die kombinatorischen Modelle bei bis zu 10000 und für sequentielle Beschreibungen je nach Art der Modellierung der Flipflops zwischen maximal 20000 und 40000 Elementen.

3.5 Evaluierungsumgebungen für parallele Simulationsverfahren

In den vergangen Jahren wurde eine Reihe von Plattformen zur Klassifikation der Leistungsfähigkeit paralleler Simulationsalgorithmen vorgestellt. Ziel war es oft, den Einfluß identifizierter und relevanter Parameter auf eine spezielle, eingeschränkte Klasse von Algorithmen (synchron, konservativ, optimistisch) für meist unterschiedliche Hardwareplattformen (Distributed- oder Shared-memory, SIMD oder MIMD) zu untersuchen. Im Unterschied zur vorliegenden Arbeit wurde eine umfassende Kombination verschiedener Klassen innerhalb eines Frameworks jedoch weniger berücksichtigt. Die folgende Beschreibung trifft in chronologischer Reihenfolge eine Auswahl repräsentativer Systeme.

- YADDES [PLH88a, PRM90a]: Einen frühen Ansatz einer Simulationssprache stellt YADDES dar. Untersuchungen mit einigen kleinen Modellen logischer Schaltkreise ergaben für die *Simulation* eines konservativen, eines optimistischen und eines synchronen Simulators stets schlechtere Ergebnisse als mit einem sequentiellen Simulator erreicht wurden.

- SPECTRUM [REY88a, RED89a, RWF89a]: Im Rahmen dieses Testbetts wurde von Reynolds und Dickens eine Umgebung geschaffen, in der die synchronisationsspezifischen Implementierungsdetails in wenigen gekapselten Modulen, den sogenannten Filtern, für eingehende und ausgehende Nachrichten sowie für die Freigabe der Simulation des nächsten Ereignisses vom eigentlichen applikationsspezifischen Simulatorcode abgeschirmt werden. Durch Austausch der

Filter kann somit nach einer Neuübersetzung auf einfache Art ein anderes Protokoll verwendet werden. Lediglich die Implementierung der neuen Filtermodule ist zu realisieren.

Mit diesem Verfahren sollte eine leichtere Überprüfbarkeit der Einflüsse der von Reynolds ermittelten Designvariablen erreicht werden [REY88a]. Leider läßt sich das Filterkonzept nicht durchgängig für beliebige Methoden realisieren [RWF89a], so daß z.B. für Time Warp auch im eigentlichen Applikationscode Änderungen zur Unterstützung des Checkpointings und der Ereignislistenverwaltung nötig werden können. In SPECTRUM wurden neben Deadlock-detection und -recovery auch die Verfahren SRADS und SRADS/LR realisiert. Ergebnisse über reine optimistische Verfahren wurden nicht publiziert. Als Rechnerplattformen wurden die Parallelrechner Butterfly GP-1000 mit Shared-memory-Architektur auf Basis vernetzter Motorola 68K-Prozessoren und ein Intel iPSC/2-Hypercube mit verteiltem Speichermodell genutzt. Veröffentlichungen zur Abgrenzung der Leistung verschiedener konservativer Verfahren sind lediglich in [RWF89a] als unklare Trends zu erkennen.

- Synapse [WLB88a]: Wagner, Lazowska und Bershad untersuchten in ihrem System Synapse auf einer Shared-memory-Architektur die konservativen Methoden Deadlock-detection und -recovery, Deadlock-avoidance und ein auf SRADS aufsetzendes Fensterverfahren (Lazy-blocking-avoidance). Es existieren in Synapse mehrere LPs pro Prozessor, die von einem Laufzeitsystem gescheduled werden. Bei Lazy-blocking-avoidance wird das Scheduling über ein Zeitfenster und Timeouts auf Simulationszeitbasis gesteuert. Das Zeitfenster schreitet mit den erhaltenen Garantien fort. Seine Obergrenze wird für die blockierten LPs von einem separaten zentralen Prozessor berechnet.

- YAWNS [NMI90a]: Mit YAWNS stellt Nicol eine Umgebung zur Entwicklung von Applikationen unter Beibehaltung eines festen synchronen aktivitätsbasierten Simulationsverfahrens vor, das ebenfalls auf einer Fenstertechnik zur Ermittlung sicherer Ereignisse basiert.

- SPEEDES [STE91a]: Steinman grenzt in diesem System die Leistung einiger fensterbasierter Verfahren mit optimistischen Zügen [STE94a], [STE93a] gegen die Basisversion des Time Warp ab. Als Plattform wurde ein Motorola 68K-basierter Hypercube und ein UNIX-Cluster verwendet.

- Einen umfassenderen Ansatz als die bisherigen Verfahren findet man erst bei Bagrodia, Chandy und Liao [BCL91a] im Rahmen von Maisie, einer Programmier- und Entwicklungsumgebung für verteilte Simulationsapplikationen. Maisie stellt neben einer eigenen Sprachdefinition zur Modellierung von Simulationsexperimenten auch einen konservativen und einen optimistischen Grundalgorithmus für die Entwickler von Applikationen zur Verfügung. Aufgrund fest definierter Schnittstellen in der Simulationssprache (ähnlich wie bei Reynolds) können aber auch eigene Simulationsalgorithmen implementiert und über Bibliotheken eingebunden werden.

  Interessant ist auch, daß Maisie auf dem sonst eher selten verwendeten Space-time-Ansatz beruht [CHS89b] und die realisierten Algorithmen Spezialisierungen ohne Nutzung der zeitlichen Parallelität darstellen (siehe Kap. 2.6.1). Über ein Relaxationsverfahren wird eine Konvergenz der Teilmodellzustände erreicht. Bei Simulationsfehlern wird zunächst versucht, diese lokal mit Fix-up-Techniken zu kompensieren. Wo dies nicht möglich ist, werden Rollbacks auf zwischengespeicherte Zustände zur Korrektur eingesetzt.

  Bagrodia entwickelte unterschiedliche Implementierungen, die zunächst auf Shared-memory-Parallelrechnern (Symult S2010: Motorola 68020 Prozessoren) liefen. Als Applikationen wer-

den insbesondere Schaltkreissimulationen auf der Switch- und Gatterebene betrachtet. Außerdem unterstützt Maisie auch Distributed-memory-Rechner [BLJ94a, BCJ95a] auf der Basis von IBMs SP1-Parallelrechner.

Speziell im Bereich der parallelen Logiksimulation von Schaltkreisen wurden ebenfalls einige Evaluierungsumgebungen entwickelt.

- Soulé [SOU92a] grenzte das Verhalten von synchronen und asynchronen konservativen Verfahren gegeneinander ab. Seine parallelen Simulatoren basieren auf dem sequentiellen ereignisgesteuerten Simulator THOR, der nach dem Fixed-delay-Prinzip arbeitet und keine Verzögerungszeiten mit dem Wert Null (Zero-delays) gestattet.

  Soulé kam zum Schluß, daß asynchrone Verfahren im Vergleich zu den synchronen Varianten stets schlechter abschneiden. Untersucht wurde die Simulation verschiedener Schaltkreise auf zwei Shared-memory-Architekturen (Encore Multimax und Stanford DASH) mit jeweils 16 Prozessoren. Die gemessenen Speedup-Werte der asynchronen Verfahren liegen bei 7-8 und 2-4 für die beiden Architekturen mit Schaltkreisen von 7000 bis 74000 Gatteräquivalenten. Die Bedeutung des Referenzwerts für die Simulation auf einem Prozessor ist nicht ganz klar, so daß es sich auch durchaus um die Laufzeiten der parallelen Version auf einem einzigen Knoten handeln könnte. Dadurch stellen die angegebenen Beschleunigungswerte obere Grenzen für die erreichbare Verbesserung dar.

  Außerdem zeigte sich die Sensibilität der Simulation für die Eigenschaften der untersuchten Schaltkreise, so daß allgemeine Optimierungen zur Berücksichtigung von Topologien und Elementeigenschaften generell keine eindeutigen Verbesserungen für alle Schaltkreise liefern konnten.

- Briner [BRI90a] untersuchte das Parallelisierungspotential von Schaltungen durch Verwendung optimistischer Verfahren auf dem Transistor- und Gate-Level (wobei der zugrundeliegende Simulator PLDVSIM auch andere Modellierungsebenen gestattet). Er betrachtete den Einfluß verschiedener Parameter u.a. Aggressive- oder Lazy-cancellation, Verwendung optimistischer Fenster unterschiedlicher Größen und Rollbacks bzw. Synchronisation auf LP- bzw. Prozessorebene (d.h. werden bei einem Rollback nur einzelne oder alle Gatter eines Prozessors zurückgesetzt).

  Im Gegensatz zu Soulé befand Briner die asynchronen (optimistischen) Verfahren den synchronen überlegen. Insbesondere bei Verwendung geeigneter Fenstergrößen, Synchronisation auf LP-Ebene und Lazy-cancellation wurden Speedups von bis zu 25 auf 32 Prozessoren erreicht. Diese Messungen trafen allerdings nur für den größten von 5 Schaltkreisen (600 bis 32000 Elemente) zu, der als Array von Prozessoren eine sehr reguläre Struktur aufweist. Die mittlere Beschleunigung betrug 8 -10 auf 32 Prozessoren eines Butterfly GP1000-Parallelrechners. In diesem Bereich unterscheiden sich die Ergebnisse für Prozessor- und LP-Synchronisation nicht stark. Der Parallelitätsverlust für Prozessorsynchronisation liegt unterhalb von 15 Prozent. Dabei spart man jedoch einiges an Speicherplatz und Synchronisationsaufwand ein.

- Bauer und Sporrer [BAS93a, SRS95a] entwickelten einen parallelen, optimistischen Schaltkreissimulator auf Gatterebene. Sie untersuchten die Beschleunigung einiger sequentieller und kombinatorischer Schaltungen aus dem ISCAS-Benchmark-Satz. Messungen erfolgten auf Distributed-memory-Architekturen (Sequent Symmetry unter Vernachlässigung der Shared-memory-Komponenten und Workstation-Cluster mit Socket-Kommunikation) und erreichten Speedups von 2-3 auf einer 5-Prozessor-Sequent sowie 5-8 auf 20 Sun Sparc2-Workstations.

Dieser parallele Simulator wurde von Schlagenhaft u.a. [SRS95a] um eine Komponente für dynamische Lastbalancierung erweitert, die die Migration von zu Simulationsbeginn festgelegten Teilclustern gestattet. Verbesserungen um 30 Prozent durch das Lastausgleichsmodul werden für Versuche mit zwei Knoten berichtet.

- Wilsey entwickelte den frei verfügbaren warped-Kernel zur Unterstützung optimistischer paralleler Simulatoren [MBW96a]. Die in Form einer Bibliothek auf Message Passing Interface (MPI) aufsetzende Software gestattet die Parallelisierung ereignisgesteuerter Simulation über festgelegte Schnittstellen ähnlich wie bei SPECTRUM und Maisie. Durch die Kommunikationsplattform MPI wird die Implementierung jedoch hochgradig portabel.

  Auf dem warped-Kernel setzt eine parallele Implementierung eines VHDL-Simulators auf (QUEST II/SAVANT). Es wird zwar von Untersuchungen mit Schaltungen von über 200000 Gattern gesprochen, konkrete Speedup-Werte werden jedoch nicht genannt [WMS98a].

- Luksch schließlich entwickelte eine Plattform zum Vergleich konservativer und optimistischer Simulationsverfahren auf einem Intel Hypercube iPSC/2 bzw. iPSC/860 und einem Netz von Sun-Workstations anhand von Schaltkreissimulationen auf Gatterebene [LUK95a]. Dabei wurden jedoch nur kleinere kombinatorische ISCAS-Schaltkreise mit weniger als 10000 Elementen untersucht und Speedups unter 4 auf 16 Prozessoren mit Time Warp erreicht. Bei konservativen Verfahren lag die Beschleunigung für die relativ kleinen Modelle unterhalb von 2.5 und oft lief die parallele Version langsamer als das sequentielle Pendant.

3.6 Universeller Einsatz paralleler Verfahren

Trotz jahrelanger Forschung hat sich für die parallele Simulation in der Industrie noch kein weit verbreitetes Anwendungsgebiet ergeben. Anfang der 90er Jahre gab es zwar einige Artikel, die die weitere Vorgehensweise bezüglich des Gebietes der parallelen Simulation speziell im Forschungsbereich untersuchten und noch zu ergründende Aspekte der PDES nannten [FUN92a, FUJ93a, FET94a, NIF94a, FUJ95a], jedoch wurde auf die mangelnde Akzeptanz für die PDES bei der Industrie nicht eingegangen.

3.6.1 Forderungen an parallele Simulationsumgebungen

Die Ursachen des zurückhaltenden Einsatzes werden lediglich an einer Stelle in der Literatur (1993) innerhalb des ORSA-Journals [ORS93a] von den führenden Forschern aus dem Bereich der PDES diskutiert. Ein Grundsatzartikel von Fujimoto [FUJ93b] zeigt einige kritische Punkte auf, die von anderen Teilnehmern dieser Diskussionsrunde aufgegriffen und kontrovers betrachtet werden. Die notwendigen Punkte zur Steigerung der Akzeptanz sind nach Meinung von Fujimoto die Bereitstellung von:

1. Simulationssprachen, die die Nutzung der parallelen Algorithmen kapseln und erleichtern, sowie

2. Applikationsbibliotheken für parallele Simulationsmodelle, die für sequentielle Simulatoren für viele Applikationsklassen bereits existieren,

3. Unterstützung von LP-übergreifenden gemeinsam nutzbaren Zustandsinformationen. Darunter fallen insbesondere in Gefechtsfeldsimulationen häufig benötigte Informationen über den Zustand von Objekten (LPs) in benachbarten Sektoren.

4. Tools zur (teil-)automatischen Parallelisierung von Simulationsmodellen, indem z.B. eine Art Farming-Konzept zur Parallelverarbeitung von Ereignissen eingesetzt wird.

Es gilt somit, dem Programmierer konventioneller Simulationsmodelle eine vertraute Umgebung anzubieten, die aber dennoch transparent die potentiellen Vorteile der Parallelverarbeitung bietet.

In den Antworten auf dieses Paper nehmen verschiedene Autoren Stellung zu den von Fujimoto formulierten Thesen. Man merkt den einzelnen Artikeln stark die jeweiligen Interessens- und Arbeitsgebiete der Autoren an.

- Abrams [ABR93a] spricht sich für eine Einbettung und einen Entwurf paralleler Simulationsmodelle in ein integriertes SE-Toolkit aus. Akzeptanz der PDES könne durch enge Kooperation eines Modellentwicklers mit einem Fachmann für PDES erfolgen. Die Vereinigung beider innerhalb einer Person sei schwierig.

- Bagrodia befürwortet stark die Weiterentwicklung von Simulationssprachen [BAG93a] und Bibliotheken, die Synchronisationsalgorithmen anbieten. Dadurch werde dem einfachen Benutzer die Last der Parallelisierung abgenommen.

- Lin spricht sich für eine einfache Schnittstelle für den einfachen Programmierer aus, die jedoch bereits effiziente Leistung durch Parallelisierung bieten muß. Daneben sollte jedoch auch ein dedizierter Zugriff auf die Details der parallelen Implementierung existieren, um dem in PDES erfahrenen Simulationsentwickler weitere Optimierungsmöglichkeiten zu bieten.

- Reynolds forderte eine Spezialhardware zur Simulationsunterstützung für gängige Arbeitsplatzrechner [REY93a], da für ihn die Leistung und nicht die Nutzbarkeit der Verfahren im Vordergrund steht.

- Unger und Cleary schließlich betonen die Notwendigkeit, das Parallelisierungspotential eines Modells vor der eigentlichen Entwicklung abschätzen zu können [UNC93a]. Dadurch können teure Fehlentwicklungen bei dem Versuch, inhärent sequentielle Modelle auf eine parallele Architektur abzubilden, vermieden werden.

Fujimoto zieht schließlich aus obigen Bemerkungen das Resümee [FUJ93c], daß neben den eigentlichen für die reine PDES-Forschung interessanten Aspekte der Optimierung von Synchronisationsverfahren ein vermutlich noch langer Weg bis zur breiten Akzeptanz der PDES als allgemein verwendbares Werkzeug steht. Dennoch sollte weiterhin versucht werden, bereits kleinere Optimierungen der Benutzbarkeit paralleler Simulatoren in den Produktionsprozeß einzubringen.

3.6.2 Umsetzung der Vorschläge

Es wurde (zum Teil schon vor 1993) versucht, die von Fujimoto formulierten Ansätze zu realisieren. Im folgenden werden einige Projekte beschrieben, die eine einfachere Nutzung der parallelen Simulation und somit eine Erhöhung der Akzeptanz zum Ziel hatten.

1. Es wurden verschiedene Sprachansätze zur Parallelisierung von Simulationsanwendungen entwickelt.

    - Die umfassendste Vorgehensweise besitzt wohl das Maisie-System [BCL91a], das eine eigene C-ähnliche Programmiersprache mit Schnittstellen zur Verwendung paralleler Synchronisationsalgorithmen bereitstellt.

- Einen ähnlichen Ansatz stellen Sim++ [BLU90a] und das im Rahmen des TeleSim-Projekts entwickelte SimKit [GFU95a] dar. Die Entwicklung der Simulationsmodelle erfolgt in einer konventionellen Programmiersprache (C / C++) und ein sequentieller bzw. paralleler Simulationskern wird beim Übersetzen einfach dazugebunden.
- Wagner, Lazowska und Bershad definierten mit Synapse eine objektorientierte Entwurfsumgebung für parallele Simulation unter Verwendung konservativer Simulationsalgorithmen [WLB88a], die allerdings nur in einer proprietären parallelen Programmierumgebung für Sequent Shared-memory-Rechner ablauffähig war.
- Mehl und Preiss gehen mit den dedizierten Simulationssprachen DSL [MEH91b] und YADDES [PRM90a] einen anderen Weg, indem sie eine spezielle Sprache zur Beschreibung von Objektverhalten (LPs), Objektinstantiierungen und Interaktionsschemata zwischen Objektinstanzen schufen.
- In Parsimony, einer Weiterentwicklung von YADDES, gibt Preiss allerdings den proprietären Ansatz wieder auf und bettet eine Simulations-API in Java ein [PRW99a].

2. Die Einbindung existierender Applikationsbibliotheken mit Simulationsmodellen wurde ebenfalls in einigen Forschungsgruppen angegangen.

   - West und Mullarney versuchten, die Programmierschnittstelle der sequentiellen Simulationsumgebung ModSim II zur Modellierung räumlicher Interaktionen (oft Gefechtsfeldsimulationen) in eine parallele Simulationsumgebung auf dem TWOS einzubetten [WEM88a].
   - Nicol und Heidelberger schlagen einen Utilarian-parallel-simulator (U.P.S.) vor [NIH96a], der auf dem prozeßorientierten Ansatz der kommerziellen, sequentiellen Simulationsumgebung CSIM [SCH86b] aufsetzt. Die in der Standardumgebung modellierten Prozesse werden in konventionellen sequentiellen Simulatoren abgearbeitet. Eine für den Benutzer transparente Zwischenschicht regelt die Synchronisation der kooperierenden Simulatoren. Es ist lediglich notwendig, die bisherigen Simulationsobjekte für die Prozeßkommunikation durch einen anderen Datentyp im Simulationsmodell zu ersetzen.

   [HEN96a] und [NIH96a] geben außerdem einen guten Überblick zu den Arbeiten in den beiden eben genannten Bereichen.

3. Die Unterstützung von gemeinsamen Variablen wurde in der Forschung bisher weitgehend ausgeklammert. Eine Ausnahme stellt die Arbeit von Mehl und Hammes dar [MEH93a]. Der Zugriff auf externe Daten erfolgt sonst weitgehend durch Anforderung über Queries oder via Updates, die der Simulator, auf dem die Variable liegt, in periodischen Abständen an interessierte Prozesse verschickt [FUJ95a]. Dadurch entsteht natürlich eine gewisse Ungenauigkeit, die mitunter zu nichtdeterministischen Effekten führt.

4. Zur automatischen Parallelisierung der PDES existieren bis auf eine in [FUJ93a] zitierte Studie keine Veröffentlichungen.

Leider spiegeln sich die Visionen der Autoren auch sechs Jahre nach ihren Prognosen immer noch nicht in verbreiteten Ansätzen wider, so daß immer noch und vielleicht sogar verstärkt die Frage nach der allgemeinen Einsetzbarkeit paralleler ereignisgesteuerter Simulation zu stellen ist.

Die vorliegende Arbeit soll zur Klärung dieser Frage beitragen und befaßt sich beispielhaft mit der Klasse der Schaltkreissimulation, deren Konzepte im nächsten Kapitel eingeführt werden.

Kapitel 4

Spezifika der Schaltkreissimulation

Schaltkreissimulation ist ein wichtiges Anwendungsfeld für Simulationen, da wie in vielen anderen Bereichen die Notwendigkeit der Überprüfung eines Entwurfs vor seiner eigentlichen Realisierung als Prototyp oder Produkt besteht. Die alternative Bezeichnung Logiksimulation hat sich aufgrund der oft anzutreffenden Modellierung von Schaltungen auf der Ebene logischer Gatterfunktionalität (UND, ODER, EXKLUSIV-ODER, usw.) eingebürgert. Dennoch bieten sich bei der Modellierung von Schaltungen wesentlich diversifiziertere Abstraktionsebenen als die reinen logischen Verknüpfungen.

In diesem Kapitel sollen die Grundbegriffe der Schaltkreissimulation, sofern sie für das weitere Verständnis notwendig sind, beschrieben und anhand der Hardwarebeschreibungssprache VHDL die Modellierungsmöglichkeiten für Entwürfe aufgezeigt werden. Insbesondere wird auf die verschiedenen Abstraktionsebenen des Schaltungsentwurfs eingegangen.

Die Simulation digitaler Schaltkreise diente der vorliegenden Arbeit jedoch in erster Linie als exemplarisches Anwendungsgebiet für die Untersuchung der Parallelisierbarkeit diskreter ereignisgesteuerter Simulation und nicht dem Ziel, eine Produktionsumgebung für kommerzielle Einsetzbarkeit zu schaffen. Die Ergebnisse lassen sich in der Regel auch auf andere Applikationsklassen übertragen. Aus diesem Grund wird bei den Untersuchungen versucht, auch von der konkreten Klasse der Logiksimulation zu abstrahieren.

4.1 Modellierungsebenen

Die Betrachtung eines Schaltungsentwurfs kann auf sehr verschiedenen Ebenen erfolgen. Typische Entwurfsumgebungen arbeiten auf einer oder mehreren der folgenden Schichten (Abb. 4.1):

- Auf oberster Ebene ist lediglich die zu erbringende Gesamtfunktionalität eines Bausteins relevant. Die z.B. mit einem ASIC oder einer CPU verknüpfte Schaltfunktion kann äußerst komplex sein. Betrachtet man beispielsweise einen Kodier-/Dekodier-Baustein, so sind die Ausgangssignale von den Eingangssignalen abhängig und werden durch eine mitunter aufwendige Berechnung erzeugt. Als Metaebene eines Chips können programmierbare Schaltungen (z.B. CPUs) über ihren Maschinenbefehlssatz simuliert werden.

- Auf der funktionalen Ebene einer Schaltung interessieren nicht die einzelnen internen Realisierungen sondern das Verhalten, das an den Schnittstellen (z.B. Pins) geboten wird. Für eine Analyse eines exakten Zeitverhaltens und die Überprüfung interner Schaltvorgänge ist diese Sicht nicht geeignet.

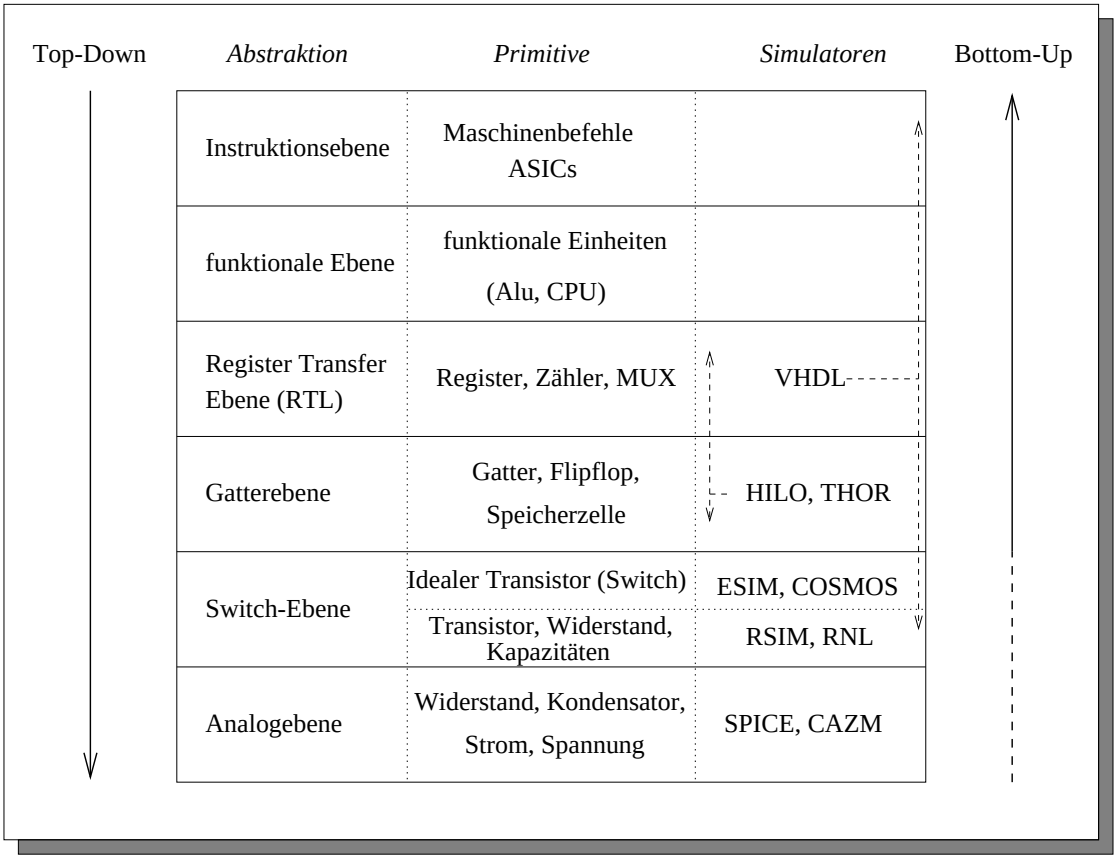


Abbildung 4.1: Modellierungsebenen

- Auf der Register-Transfer-Ebene (RT-Level) werden elementare Bausteine zum Aufbau eines Schaltkreises verwendet. Komponenten sind hier Register, Multiplexer und Demultiplexer, Multiplizierer, Addierer usw. Bei der Simulation mit RT-Bausteinen interessiert die Interaktion der einzelnen Teile. Es ist zu überprüfen, ob die zeitliche Verarbeitung der Ausgangs- und Eingangssignale der Baugruppen korrekt miteinander verzahnt ist. Die RT-Komponenten selbst werden dabei oft nur von ihrem nach außen hin sichtbaren Zeitverhalten und ihrer Funktionalität gesehen. Es sind somit quasi Schaltkreise im Kleinen.

- Die RT-Komponenten setzen sich wiederum aus elementaren logischen Schaltungsobjekten, den Gattern, zusammen. Die Gatterebene bietet als erste Verfeinerung eine Analysemöglichkeit für komplexe Zeitverhalten [CHA85a, HIL85a, ABC88a]. Die Schaltzeiten der Gatter können gezielt angegeben werden und somit z.B. Hazards oder Races in einem Entwurf erkannt und vermieden werden. Das Auftreten solcher meist unerwünschten Effekte könnte sonst beim Zusammenschalten elementarer (RT-)Elemente zu unliebsamen Komplikationen führen. Aus diesem Grund ist es in der Regel auch nötig, daß man die Simulation großer Schaltungen auf dieser oder einer noch niedrigeren Ebene durchführt. Hierarchische Simulationen schließen in der Regel nur gröbste Fehlverhalten aus. Die detailierte Untersuchung der Schaltungen hat auf einer der unteren Ebenen zu erfolgen.

- Auf dem Switch-Level [BBB87a] wird auf einer abstrakten Ebene die idealisierte Schaltfunktion der einzelnen Transistoren betrachtet. Je nach verwendeter Basistechnologie besteht ein

Gatter aus einer unterschiedlichen Anzahl von Transistoren. Ob ein Modellierer Kapazitäten und Zustand der getriebenen Transistoren berücksichtigt, liegt in seinem Ermessen. Es ist im Gegensatz zur Gatterebene jedoch eine weiter verfeinerte Untersuchung des Zeitverhaltens durch eine exaktere Spezifikation der Schaltvorgänge möglich. Die idealisierten Schalter der Switchebene besitzen als unterste digitale Schicht immer noch diskrete Zustände. Bei Berücksichtigung unterschiedlicher Kapazitäten durch getriebene Transistoren schlägt sich dies in wechselnden Schaltzeiten (Variable-delay) nieder.

- Auf der niedrigsten Ebene (Schaltkreis) existieren keine diskreten Signalzustände mehr. Die Schaltkreise werden durch Transistoren mit spezifischen Kennlinien für Strom, Spannungen und Kapazitäten repräsentiert. Diese sind je nach verwendeter Basistechnologie (CMOS, bipolar, ECL) sehr verschieden. Die Simulation erfolgt auf dieser Ebene entweder in sehr fein gerasterten zeitgesteuerten Verfahren oder komplett durch Lösung von Differentialgleichungen für die analogen elektrischen Schaltungen. Ziel dieser extrem feingranularen Simulation ist die Überprüfung des zeitlichen Verhaltens einer Schaltung für eine konkrete Realisierung mittels einer bestimmten Fertigungstechnik.

Verbreitete sequentielle Simulatoren für die unterste Ebene sind z.B. SPICE von der Universität Berkeley [JQN92a] und PSPICE [ORC99a]. Außerdem besteht die Möglichkeit, mit diesen Werkzeugen gemischte analoge und digitale Modelle auf den untersten beiden Ebene zu simulieren. Diese Koppelung beider Techniken wird als *Mixed-mode-* oder *Mixed-signal*-Simulation bezeichnet.

Für die Simulation der darüberliegenden Schichten existiert ebenfalls eine Vielzahl sequentiell arbeitender, kommerzieller Produkte. Die vom amerikanischen Verteidigungsministerium (DoD) zum Standard erhobene Hardwarebeschreibungssprache VHDL (Very High Speed Integrated Circuit HDL) [VHD87a] bietet eine Schnittstelle zur Realisierung hierarchischer Modelle auf allen digitalen Entwurfsebenen. Außerdem soll Analog HDL (AHDL) auf der analogen Schaltkreisebene ähnliche Spezifikationsmöglichkeiten wie VHDL bieten. Von den Standardisierungsgremien der IEEE wird an VHDL-Erweiterungen für die Unterstützung der Analogsimulation gearbeitet (VHDL-AMS, analog and mixed-signal) [IEE99a]. XSPICE vom Georgia Institute of Technology erweitert die Koppelungsmöglichkeiten von analogen und digitalen Schaltungen auf der Transistorebene [INT98a].

Der Entwurfsprozeß kann dabei wie bei der Entwicklung von Softwarekomponenten nach dem Top-down- oder dem Bottom-up-Verfahren erfolgen. Von der Gesamtfunktionalität eines Schaltkreises ausgehend kann somit beim Top-down-Design in Verfeinerungsschritten das endgültige Produkt erstellt werden. In der Regel wird man aber auch bereits existierende Bibliotheken mit elementaren Bausteinen oder RT-Primitiven besitzen, die man zu komplexeren Schaltungen zusammensetzt, so daß sich eine Kombination aus Bottom-up für den seltenen Fall eine Neuentwicklung von Basiskomponenten mit der Identifizierung von Standardkomponenten bei der Zerlegung in Top-down-Richtung ergibt.

4.2 Modellkomponenten

Bei der Modellierung von Schaltkreisentwürfen werden ähnliche Begriffe wie zur Beschreibung von Graphen verwendet, die jedoch leicht unterschiedliche Bedeutung aufweisen. Da wir bei der Aufteilung von Schaltungen für die parallele Simulation Graphenpartitionierungsalgorithmen einsetzen, sollen hier die Begriffe gegeneinander abgegrenzt werden, damit es später nicht zu Mißverständnissen kommt.

Definition 1 *Ein Objekt bezeichnet ein Element einer Schaltung, dem eine entsprechende Schaltfunktion zugeordnet ist. Es besitzt in der Regel einen oder mehrere Ein- und Ausgänge, die die Eingabewerte der Schaltfunktion liefern und Ausgabewerte aufnehmen. Je nach Abstraktionsebene wird ein Objekt auch als Transistor, Switch, Gatter, Modul oder IC bezeichnet. Ein Objekt besitzt für jeden diskreten Zeitpunkt einen bestimmten Wert, der sich alleine aus den zu diesem Zeitpunkt anliegenden und intern eventuell gespeicherten Informationen bestimmt.*

Das Zusammenschalten von Objekten erfolgt durch Signale. Wir beschränken uns hier auf den Fall der diskreten Simulation zeitdiskreter Größen.

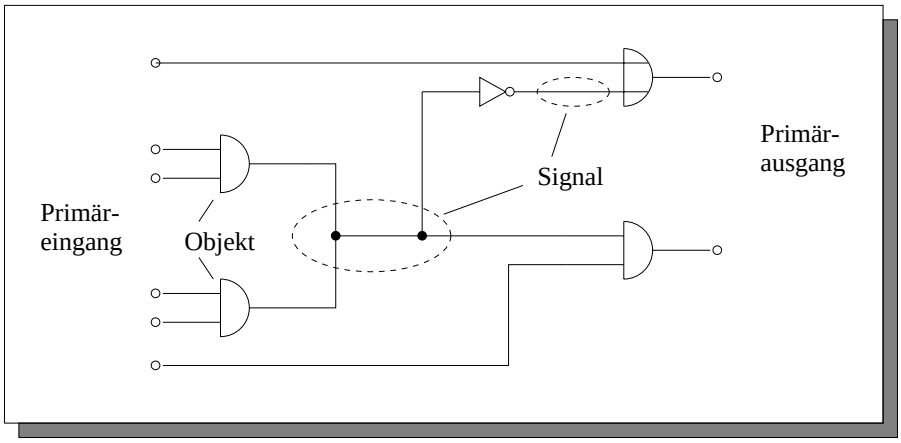


Abbildung 4.2: Modell mit Objekten und Signalen auf Gatterebene

Definition 2 *Ein Signal stellt eine zeitliche Abfolge diskreter Wertänderungen zu diskreten Zeitpunkten dar. Es verbindet die Ausgänge von null oder mehreren Objekten mit den Eingängen von null oder mehreren Objekten. Innerhalb eines Simulationsmodells wird ein Signal durch einen Knoten (oder Node) repräsentiert, der die zeitliche Entkoppelung der angeschlossenen Objektausgänge von den mit ihm verbundenen Eingängen garantiert. Deshalb werden oft auch die Ausdrücke "Knoten" oder "Node" zur Bezeichnung eines Signals verwendet. Die Menge aller an einen Knoten angeschlossenen Leitungen (Eingänge und Ausgänge) bildet ein Netz, dem ein Gewicht entsprechend seiner Größe zugewiesen werden kann.*

Definition 3 *Ein Signal, das keine Objektausgänge besitzt, heißt Primäreingang. Verfügt ein Knoten über keine angeschlossenen Objekteingänge, so ist er ein Primärausgang.*

Die hier beschriebene Trennung der Ausgangs- und Eingangssignale hat eine in der Simulationstechnik begründete Ursache. Die Auswertung der Objekte erfolgt in einer sequentiellen Reihenfolge. Änderte nun die Auswertung eines Objekts sein Ausgangssignal (für einen späteren Zeitpunkt) sofort, und wird das Signal von einem noch nicht bearbeiteten anderen Objekt als Eingang verwendet, so geht der ursprüngliche Wert verloren und die Auswertung des zweiten Objekts arbeitet mitunter mit falschen Eingabedaten.

Deshalb werden in einem zweistufigen Verfahren zunächst alle Objekte mit den aktuell vorliegenden Eingabewerten evaluiert und die Ausgabedaten als temporäre Zwischenwerte in den Nodes abgelegt. Dabei kann anhand einer Entscheidungsfunktion auch ermittelt werden, welcher endgültige Signalwert verwendet werden soll, wenn gleichzeitig mehrere Objekte dieses Signal steuern (treiben). Gleichzeitig werden auch Signaländerungen für verschiedene Zeitpunkte in der Zukunft (sog.

projektierte Signalverläufe) von den Nodes verwaltet und in Abhängigkeit vom Zustand der treibenden Objekte angepaßt.

In der zweiten Phase werden die geänderten Signale dann an die Objekte als neue Eingabedaten propagiert und stehen für die nächste Auswertung bereit (Abb. 4.3).

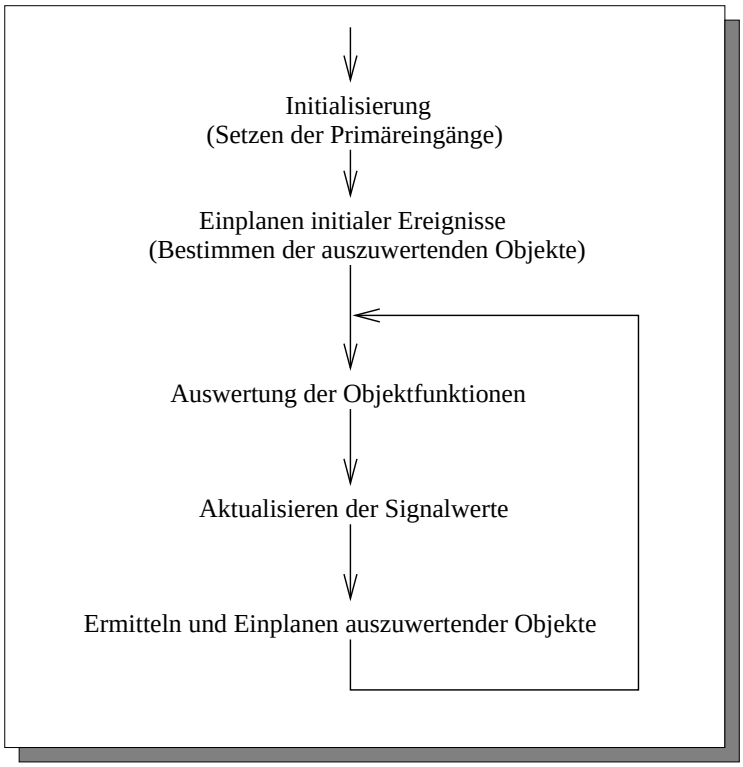


Abbildung 4.3: Zweistufige Auswertung von Objekten und Signalen

4.3 Timing-Modelle

Die Modellierung der Simulationszeit wurde bereits im letzten Kapitel eingeführt (Unit-, Fixed- und Variable-delay). Allerdings werden bei der Simulation von Schaltkreisen oft idealisierte Elemente mit einer Schaltzeit von Null Zeiteinheiten eingesetzt. Die Verzögerung realisiert in solchen Fällen ein nachgeschaltetes Delay-Glied.

Die Knoten des Modells müssen somit in der Lage sein, Ereignisse, die den gleichen Zeitstempel besitzen, aber dennoch kausal voneinander abhängen, zu unterscheiden. Als Lösung wird eine Kombination der zweistufigen Auswertung von Objekten und Signalen, die ja implizit schon eine Trennung kausal abhängiger Ereignisse bietet, mit einem erweiterten Zeitstempelkonzept [MEH91a] eingesetzt.

Ein Zähler registriert in einer zweiten Komponente der Simulationszeit die Iterationsanzahl innerhalb gleicher Zeitstempel. Bei identischer Simulationszeit wird der Zähler erhöht. Andernfalls wird der Zeitstempel auf die neue Zeit gesetzt und der Zähler mit Null belegt. In VHDL wird dieses Konzept als Delta-delay bezeichnet [VHD87a, CHW91a].

4.4 VHDL

Viele Hardwarebeschreibungssprachen (HDLs) gestatten eine strukturelle Beschreibung der damit zu modellierenden Schaltungen. Es besteht die Möglichkeit, Objekte nach ihrem Typ und Eigenschaften zu klassifizieren, Schaltfunktionen über Parametrisierung festzulegen und die Topologie der Schaltung durch Verknüpfung über Signale zu bestimmen. Der Entwurf erfolgt in der Regel auf speziellen CAD-Workstations unter Verwendung einer graphischen Oberfläche, mit der man Objekte plazieren und verbinden kann. Parameter können über Popups eingestellt werden. Die Generierung des Modells und Layouts erfolgt dann automatisch aus der graphischen Beschreibung der Schaltung. Das CAD-System verbirgt die eigentliche Modellbeschreibungssprachen, die für den Entwickler oft nicht relevant ist, hinter Objektbibliotheken mit vorgefertigten Schaltelementen für unterschiedliche Fertigungstechnologien.

VHDL wurde als Standard geschaffen, um eine einheitliche Schnittstelle für Entwicklungen für das amerikanische DoD zu bieten. Der Zwang, der den Entwicklern mit dieser Sprache auferlegt wurde, hat einerseits zu einer relativ schnellen Verbreitung geführt. Andererseits ist VHDL aber aufgrund seiner Komplexität nicht unumstritten.

4.4.1 Sprachkonzepte

Mit VHDL wurde eine vollständige hochsprachliche Programmiersprache geschaffen, die über Datentypen, Variablen, Modularisierung durch Packages, Kontrollstrukturen sowie eine Schnittstelle zur C-Programmierung verfügt und Zuweisungen um eine Zeitkomponente erweitert, die relativ zum aktuellen Zeitpunkt angibt, wann die Wertzuweisung erfolgen soll. Verschiedene mehrwertige Logiken werden zur Modellierung von Signalen angeboten. Standard ist eine 9-wertige Logik mit Erweiterungen für schwache, undefinierte und hochohmige Signale[1].

Neben der statischen, rein strukturellen Darstellung, wie man sie auch in anderen HDLs findet, gestattet VHDL sowohl eine dynamische Verhaltensbeschreibung, die einen beliebig komplexen Entwurf ermöglichen, als auch eine logische Beschreibung einer Schaltung anhand boole'scher Gleichungen in Form eines Datenflußmodells.

Die Definition von Schnittstellen und ihre Realisierung ist durch ein *Entity-/Architecture*-Modell getrennt. Ein Entity stellt die statische Schnittstelle eines Objekts mit Angabe von Eingangs- und Ausgangssignalen und Parametern für die Festlegung des zeitlichen Verhaltens dar.

Für jedes Entity können verschiedene Architekturen verwendet werden. Die konkrete Zuordnung einer Architekturimplementierung erfolgt im Simulationsmodell bei der Instantiierung der Objekte.

Die Architektur selbst ist entweder eine strukturelle Beschreibung, die ggf. hierarchisch aus weiteren Entities aufgebaut die Verschaltung der einzelnen Bauelemente über Signale definiert, eine Sammlung boole'scher Gleichungen (in Form von VHDL-Signalzuweisungen) oder ein VHDL-Prozeß (`process`-Statement), der in seiner Syntax einem C-Programm ähnelt und das dynamische Verhalten beschreibt. Kontrollflußkonstrukte, Variablen und Einbettung von C-Programmteilen können in VHDL nur innerhalb von Prozessen verwendet werden.

Außerdem verfügen VHDL-Prozesse über einen `wait`-Befehl, um auf den Eintritt bestimmter Ereignisse zu warten. Die Abarbeitung der Befehle eines Prozesses erfolgt sequentiell in einer den gesamten Block implizit umgebenden Endlosschleife. Bei Erreichen eines Wait-Statements blockiert ein solcher Prozeß und setzt nach Erfüllung der Bedingung (Signalwechsel, Timeout) die Bearbeitung mit dem nachfolgenden Befehl fort. Bei der Initialisierung läuft ein Prozeß ebenfalls bis zum ersten Wait-Befehl. Existiert kein solcher, bleibt der Simulator in der Endlosschleife hängen. Alter-

[1] Für spezielle Modellierungsfälle ist auch eine 46-wertige Logik vorgesehen [COE89a].

nativ zum Wait-Statement kann auch eine Menge von Signalen in einer *Sensitivity*-Liste angegeben werden, die vor Beginn eines neuen Schleifendurchlaufs auf Signaländerungen überprüft wird. Erst wenn ein solcher auftritt, wird die Bearbeitung fortgesetzt.

Ein VHDL-Modell muß zur Simulation, insbesondere wenn es C-Erweiterungen enthält, zunächst einmal übersetzt werden, um eine Zuordnung von Entities und Architekturen durchzuführen sowie Instanzen der Modellkomponenten zu generieren. Die Ausgabe des VHDL-Kompilers besteht bei rein strukturellen Beschreibungen meist aus eine Liste elementarer Objekte (z.B. Gatter), die von einem generischen Simulator bearbeitet werden kann. Es besteht dabei aber bei größeren VHDL-Systemen auch die Möglichkeit, Komponenten auf unterschiedlichen Hierarchieebenen zu verwenden.

Soll dynamisches Verhalten simuliert werden, so erzeugt der VHDL-Kompiler zusätzlichen Programmcode, der die Abarbeitung der VHDL-Prozesse in (C-)Funktionen umsetzt. Die generierten Programmteile müssen für jedes Modell (und nach jeder Änderung) mit einem generischen Simulatoranteil gebunden werden, wodurch jedes Simulationsmodell ein eigenes Simulationsprogramm erhält.

Diese Vorgehensweise ist neben dem programmiersprachlichen Ansatz auch ein Hauptkritikpunkt der Verfechter rein interpretativer Beschreibungen, wie sie bei Analogschaltungen im SPICE-Modell [UCB97a] weitgehend vorliegen. Ein Entwickler sollte nach ihrer Meinung mit der konkreten Realisierung der Schaltelemente eigentlich nicht belastet werden sondern einen Baukasten mit von ihm benötigten Elementen sowie eine Möglichkeit zum einfachen Zusammenstecken an die Hand bekommen.

Dem ist jedoch zu entgegnen, daß z.B. auch in SPICE der Bedarf zur Modellerweiterung gesehen wird, und neue Modellkomponenten über eine nachträglich zum System hinzugefügte C-Schnittstelle integrierbar sind. In VHDL ist diese Möglichkeit adhoc vorgesehen und ermöglicht insbesondere die leichte Erstellung neuer Bausteinbibliotheken.

4.4.2 Struktur eines VHDL-Simulators

VHDL-Simulatoren arbeiten nach dem ereignisgesteuerten Prinzip. Die Objekte dienen dabei als Ereignislieferanten. Innerhalb der VHDL-Modellbeschreibungen finden sich als Basisfunktionen die Berechnung und Zuweisung neuer Werte an Signale (via <=-Operator). Die Zuweisungen werden jedoch aufgrund der oben bereits beschriebenen Trennung von Ausgangs- und Eingangssignalen durch die Nodes nicht sofort ausgeführt, sondern als Ereignisse und Transaktionen mit dem zugehörigen Zeitstempel und neuem Signalwert gesammelt. Ein Ereignis repräsentiert den Wechsel des Signalwerts. Generiert eine Zuweisung an ein Signal wieder den bereits vorhandenen Wert, so wird das als Transaktion bezeichnet [ADM93a]. Die PDES modelliert beide VHDL-Ereignistypen als Simulationsereignis.

Nach Auswertung aller Objekte, für die sich Eingangssignale geändert haben, erfolgt die Verarbeitung der zwischengespeicherten Signaländerungen. Es ist zu beachten, daß für einen VHDL-Simulator keine bestimmte Auswertungsreihenfolge für Objekte und Signalzuweisungen vorgegeben wird (z.B. anhand einer fortlaufenden Beschreibung im Modell), da die Nodes ja für eine kausale Entkoppelung sorgen. Innerhalb der VHDL-Prozesse werden Anweisungen jedoch sequentiell abgearbeitet und auch die Zuweisungen an die nur intern sichtbaren Variablen[2] erfolgen unverzüglich.

Die zwischengespeicherten Ereignisse werden anschließend in eine Liste möglicherweise bereits für andere Zeitpunkte (oder aber auch für dieselbe Simulationszeit) gespeicherter Ereignisse des

[2]Variablen sind in VHDL ähnlich wie in einer konventionellen Programmiersprache als Speicher für Zwischenberechungen vorgesehen.

jeweiligen Knotens eingetragen. Dabei können Knoten zwei verschiedene Typen von Signalen repräsentieren. Man unterscheidet generell beim Schaltungsentwurf zwischen Signalen, die alle Eingangsänderungen direkt (ggf. mit einer Verzögerung) durchschalten und solchen, bei denen ein Wert für eine bestimmte Mindestdauer stabil anliegen muß, damit eine Schaltfunktion ausgelöst wird. Die erste Variante (*Transport-delay*) wird beispielsweise zur Modellierung von reinen verzögerungsbehafteten Übertragungsleitungen verwendet, während die zweite (*Inertial-delay*) die Trägheit von Schaltelementen nachbildet.

Ein Transport-Signal übernimmt alle projektierten Ereignisse in entsprechender zeitlicher Reihenfolge in die Treiberliste für seine Ausgangssignale (diese sind für die angeschlossenen Objekte aus deren Sicht Eingangssignale). Wird ein neues Ereignis eingeplant, so werden alle späteren Ereignisse gelöscht, da stets der aktuellste berechnete Wert als der korrekte angenommen wird. Bei einem Signal, daß zum selben Zeitpunkt von zwei Ausgängen getrieben werden soll, kommt eine Resolutionsfunktion zum Einsatz.

Die Bearbeitung träger Signale ist etwas komplizierter. Ändert sich ein Signalwert, so werden alle bis dahin eventuell eingeplanten (auch die zeitlich vor ihm liegenden) Ereignisse hinfällig, da ein neuer Wert dynamisch berechnet wurde und alle bereits vorhandenen Signalverläufe unter der Annahme geplant waren, die Eingaben des Nodes blieben stabil. Alle existierenden Ereignisse werden also aus der projektierten Ereignisliste gestrichen und durch das neue Ereignis ersetzt. Es besteht weiterhin die Möglichkeit, eine Folge von Signalwechseln in eine einzige VHDL-Signalzuweisung einzubetten. Jeder Wert entspricht dann einem neuen Ereignis, die aber (nur bei der Einplanung) logisch wie ein einziges Event behandelt werden. Die Abbildungen 4.4 und 4.5 geben jeweils ein Beispiel für die Abarbeitung von VHDL-Signalzuweisungen mit den unterschiedlichen Modellen (aus [ROS92a]).

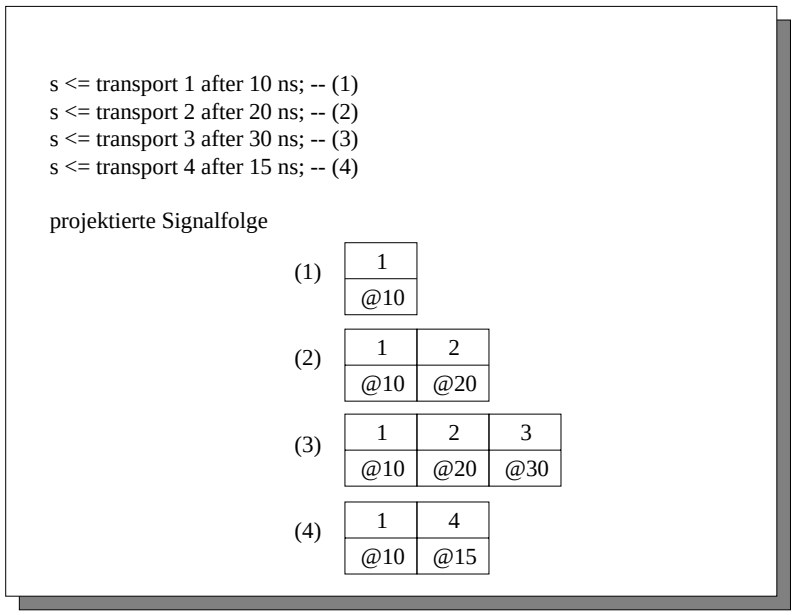


Abbildung 4.4: Ereigniseinplanung mit Transport-Signalen

Allerdings gibt es beim Löschen der existierenden Ereignisse bei Inertial-Signalen eine Ausnahme: Ist der neue Signalwert identisch mit dem unmittelbar vor ihm abzuarbeitenden, und ist die Verzögerung des neuen Events (relativ zum aktuellen Zeitpunkt) größer als die des bereits eingeplanten Ereignisses, so subsumiert das neue Ereignis das bereits existierende. Aus diesem Grunde

bleibt dieses Ereignis mit kleinerem Trägheitselement zusammen mit dem neuen Ereignis erhalten. Somit genügt die schwächere zeitliche Bedingung bereits, um den Schaltvorgang auszulösen. Das zweite Ereignis repräsentiert in der Terminologie von VHDL eine Transaktion, da es keinen Signalwechsel bewirkt.

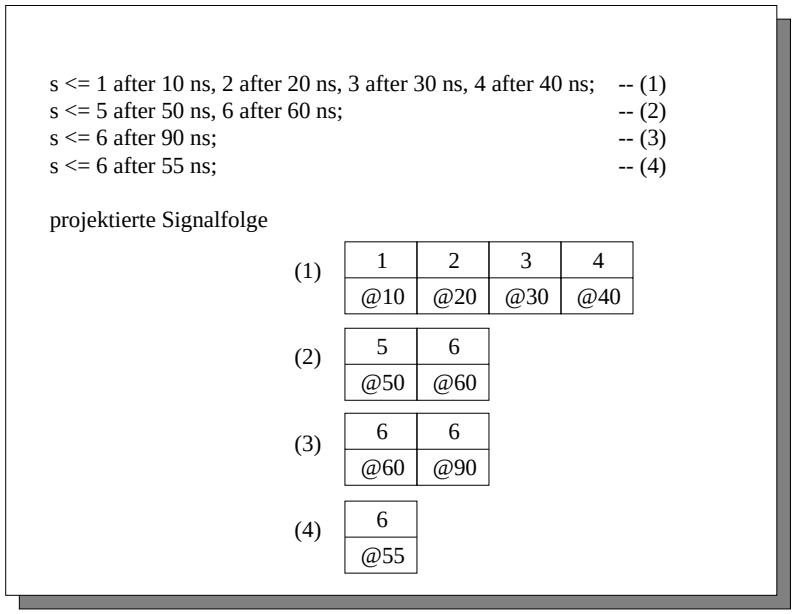


Abbildung 4.5: Ereigniseinplanung mit trägen Signalen

Der Standardtyp von Signalen in VHDL ist inertial. Die Berechnung neuer Signalfolgen erfolgt somit, indem zunächst stets die Aktionen für Transport-Verzögerung ausgeführt werden, da dieser Vorgang für beide Signaltypen zutrifft. Bei trägen Signalen werden danach zusätzlich die vorausgehenden Zuweisungen mit der oben beschriebenen Ausnahme gelöscht, wenn es sich bei dem neuen Ereignis um eine Transaktion handelt.

4.5 Parallelisierung von Schaltkreissimulatoren

Aktuelle Ansätze und Probleme bei der Parallelisierung der Schaltkreissimulation werden in [MEI93a], [BBC94a] und [CHA95a] beschrieben. Zu den Faktoren, die die Leistung eines parallelen Simulators beeinflussen, zählen Zeitmodellierung im Simulationsmodell, Modellstruktur (Topologie), Zielarchitektur, Partitionierung und Synchronisationsverfahren. Es existieren Untersuchungen zu den verschiedenen Aspekten für unterschiedliche Simulationstechniken (zeitgesteuert, synchron und asynchron), unterschiedliche Partitionierungsverfahren und auf verschiedenen Hardwareplattformen. Jedoch sind die Ergebnisse aufgrund sehr variierender Detailsichten und der Kombination einzelner Aspekte ebenso unvergleichbar wie bei Simulationsexperimenten in anderen Applikationsbereichen (z.B. Warteschlangennetze). Die einzelnen leistungsbezogenen Problembereiche werden in den nachfolgenden Kapiteln detailiert untersucht. Ihre Ähnlichkeit mit den Fragestellungen in anderen Applikationsbereichen der parallelen Simulation gestattet dann auch eine Verallgemeinerung trotz anfänglicher Beschränkung auf den Bereich der Logiksimulation.

4.5.1 Detailfragen bei der Parallelisierung

An dieser Stelle wird jedoch noch auf einige Randprobleme bei der Realisierung paralleler Schaltkreissimulatoren hingewiesen.

4.5.1.1 Verteilung der Nodes

Bei der Aufteilung des Schaltkreises in disjunkte Submodelle, die zur Simulation auf die verwendeten Prozessoren verteilt werden, erfolgt in erster Linie die Zuordnung von Objekten zu den Prozessoren. Die Anordnung der Nodes, die die einzelnen Objekte miteinander verbinden, ergibt sich daraus zwangsläufig. Allerdings existieren dabei zwei Aspekte, die berücksichtigt werden müssen.

Ein Knoten kann prinzipiell auch von mehreren Objekten getrieben werden. Idealerweise sollten sich alle diese Objekte auf demselben Simulator befinden, damit die Entscheidungsfunktion leichter realisiert werden kann. Somit werden die Datenstruktur und Funktionalität eines Knotens, die die Berechnung der projektierten Signalverläufe und die Ereignislistenfunktion realisieren, zweckmäßigerweise auf dem Prozessor angesiedelt, auf dem seine Treiberobjekte liegen. Diese Datenstruktur stellt sozusagen den Master eines Nodes dar.

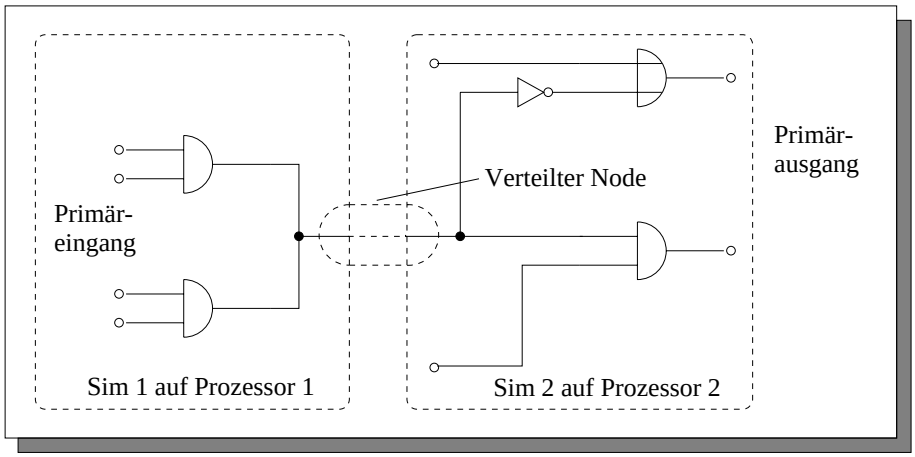


Abbildung 4.6: Verteilung von Nodes auf mehrere Prozessoren

Weiterhin kann jeder Knoten selbst wieder mehrere Objekte treiben. Um eine Aufteilung in Submodelle nicht gänzlich zu verhindern, ist es deshalb nötig, für Eingangssignale von Objekten, die auf verschiedenen Prozessoren simuliert werden sollen, eine duplizierte Ereignishaltung einzuführen. Die Ereignissequenz, die der Master des zugehörigen Knotens berechnet, wird auf den abhängigen Prozessoren gespiegelt. Die Konsistenz wird über den Versand von Ereignisnachrichten sichergestellt. In verteilten Systemen mit gemeinsamem Speicher existiert prinzipiell dasselbe Problem, es verlagert sich lediglich in Richtung Konsistenzhaltung von Caches bzw. die Organisation des Zugriffs auf nicht-prozessorlokale Datenstrukturen.

4.5.1.2 Zustandssicherung von VHDL-Prozessen unter Time Warp

Beim Checkpointing treten unter Time Warp für VHDL-Prozesse besondere Probleme auf. Insbesondere enthält eine volle VHDL-Implementierung auch Funktions- und Prozeduraufrufe und die Wait-Statements können an beliebiger Stelle im Code auftreten. Bei einem Checkpoint ist somit die Speicherung des Stacks sowie die Angabe der aktuellen Position im Code notwendig, damit

im Falle eines Rollbacks der korrekte Zustand wiederhergestellt werden kann. Bislang wurde kein Checkpointing von VHDL-Prozessen in DVSIM realisiert. Unser Subset von VHDL unterstützt allerdings auch keine Funktionen und Prozeduren. Durch Beschränkung der Wait-Statements auf den Prozeßanfang oder das -ende ließe sich eine einfache Variante schaffen. Die Diskussion genereller Checkpointinglösungen für optimistische Simulatoren wird zum Teil in ADVISE! [HOC97a], einer Simulationsumgebung für einen parallelen VHDL-Simulator aus dem ESPRIT-Projekt CHESS (A MIPS Cruncher for Concurrent Heterogenous Simulation Systems), angesprochen.

4.5.2 Parallelisierungsprojekte von VHDL-Simulatoren

Berichte über die Parallelisierung von VHDL-Simulationen sind bislang wenig in der Literatur anzutreffen. Die Ursachen liegen einerseits in der Komplexität von VHDL, die einen kompletten Neuentwurf eines Simulatorkerns, der anschließend parallelisiert werden kann, fast unmöglich macht. Andererseits sind sequentielle VHDL-Simulatoren, auf deren Basis man parallele Simulatoren entwickeln könnte, aufgrund der wirtschaftlichen Bedeutung nicht oder nur gegen sehr hohe Kosten als Quellcode verfügbar.

Die vorhandenen Parallelisierungsprojekte wurden entweder von Forschungseinrichtungen durchgeführt, die mit dem amerikanischen Militär zusammenarbeiten und dadurch Zugriff auf VHDL-Simulator-Quellcode haben, oder sie realisieren als Eigenentwicklung nur einen Teil des vollen VHDL-Sprachumfangs.

- Vellandi und Lightner entwickelten einen Simulator, der VHDL-Modelle auf einer Connection Machine CM-2 in synchroner Weise bearbeitete. Insbesondere wurden die Simulationshilfsfunktionen parallelisiert [VEL92a]. Die CM-2 verfügt über eine große Anzahl von Prozessoren, so daß nach Aussage der Autoren viele Objekte wirklich parallel ohne Scheduling oder Mapping abgearbeitet werden können. Diese Behauptung steht jedoch im Widerspruch zu den Ergebnissen von Bailey und Snyder [BAS88a], wonach bei synchroner Abarbeitung nur geringe Teile einer Schaltung gleichzeitig aktive Ereignisse haben, und der Tatsache, daß von Vellandi keine höheren Speedups als 11 erreicht wurden (eine CM-2 hat in der Regel zwischen 8192 und 65536 Prozessoren).

- Ashenden, Detmold und McKeen untersuchten in einer simulierten parallelen Umgebung mit mehreren Threads das Parallelisierungspotential von VHDL-Verhaltensbeschreibungen (Processes) anhand eines zeitgesteuerten Algorithmus [ADM93a]. Konkrete Speedups werden jedoch nicht genannt. Allerdings zeigte sich, daß die Abbildung jedes VHDL-Prozesses auf einen eigenen LP aufgrund der feinen Granularität nicht realistisch ist, sondern ein Clustern mehrere Objekte benötigt wird, um den Schedulingaufwand für die LPs zu reduzieren und einen Gewinn durch Parallelisierung zu erreichen.

- Koch und Tavangarian beschreiben in [KOT94a] einen parallelen VHDL-Simulator mit konservativem Synchronisationsverfahren, der auf einem Netz mit Workstations unter PVM arbeitet. Für mittlere Modelle ab 10000 Objekte wurde eine Beschleunigung von 2.7 auf 5 Workstations beobachtet.

- [WAG95a] beschreibt ein konservatives Verfahren zur VHDL-Simulation mit dynamischer Lookaheadbestimmung zur Vermeidung von Deadlocks und Nullnachrichten auf Basis von künstlich in das Modell eingefügten Pseudokomponenten. Diese Vorgehensweise wirkt etwas konstruiert.

- Krishnaswamy und Banerjee entwickelten einen optimistischen parallelen VHDL-Simulator [KRB96a]. Jeder VHDL-Prozeß wurde in einen eigenen LP eingebettet, der als Actor abläuft. Messungen erfolgten auf Shared- und Distributed-memory-Umgebungen (SparcServer 1000, Intel Paragon) mit kombinatorischen ISCAS-Schaltungen [BRF85a] und weniger als 300 VHDL-Prozessen. Die Basis der Speedup-Werte ist nicht klar ersichtlich. Die langen Laufzeiten der kleinen Modelle auf einem Prozessor legen jedoch die Vermutung nahe, daß dazu die parallele Version mit Actor-Scheduling auf einem einzigen Prozessor verwendet wurde.

- Wilsey entwickelte auf Basis einer vollen Implementation von VHDL den optimistischen parallelen Simulator QUEST auf Basis der Synchronisationsbibliothek *warped* [WMS98a] für Parallelrechner (IBM SP1) und Workstations.

- Einen eher analytischen Ansatz beschreiten Costa u.a. [CDF94a], indem sie den zu erwartenden Speedup bei einem VHDL-Simulator über die Auswertung von Trace-Files der beobachteten tatsächlichen Ereignisreihenfolge in einem sequentiellen Simulator nachträglich berechnen. Dazu verwenden sie die Simulation eines verteilten Simulators mit parametrisierbaren Größen wie Kommunikationsdauer, Rollback-Overhead und zugrundeliegende Kommunikationstopologie.

Das hier entworfene Framework DVSIM entstand parallel zu den Entwicklungen von Bauer, Sporrer und Krodel [BSK92a], Luksch [LUK93a] und Koch und Tanvangarian [KOT94a]. Es hat sich als Ziel die Untersuchung einer breiten Palette von Einflußfaktoren gesetzt. Insbesondere zählen dazu die im nächsten Kapitel beschriebenen Partitionierungsverfahren, Parallelisierungspotentiale und Leistungsbewertung (Kapitel 7) sowie Experimente mit typischen Vertretern der klassischen parallelen asynchronen Simulationsverfahren (Kapitel 8).

Kapitel 5

Partitionierungs- und Mappingverfahren

Bei den parallelen asynchronen ereignisgesteuerten Simulationsverfahren arbeitet in der Regel jeder der beteiligten Simulatoren auf einem eigenen Teilmodell, das von denen der anderen bis auf gemeinsame Signale disjunkt ist. Grundelemente der Verteilung sind bei der Schaltkreissimulation die Objekte, die die Schaltungselemente repräsentieren. Die Verteilung der Signale (Knoten) ergibt sich nach den im letzten Kapitel genannten Kriterien, die ein Signal dem Prozessor zuordnen, auf dem seine treibenden Objekte plaziert werden. Die Vereinigung aller Teilmodelle ergibt wieder das Gesamtmodell. Ist P die Menge aller Objekte, so gilt für eine Partitionierung $P_1, \ldots, P_n$ in n Submodelle:

$$P_i, P_j \subseteq P, 1 \leq i, j \leq n, i \neq j : P_i \cap P_j = \emptyset$$

und

$$\bigcup_{i=1}^{n} P_i = P.$$

Unter eine Partitionierung verstehen wir also die Aufteilung eines Gesamtmodells in Teilmodelle. Beim *Mapping* dagegen geht es darum, die gefundenen Partitionen einem konkreten der zur Verfügung stehenden Prozessoren zuzuordnen. Es ist dabei zu beachten, daß diese Aufgabe selbst im Falle von identischer Prozessoranzahl k und Partitionsanzahlen $n = k$ nicht kanonisch durch beliebige Zuordnung eines Submodells an einen Rechner erfolgen kann. In der Regel sind in Abhängigkeit von Topologie des Kommunikationsnetzes oder Leistungsfähigkeit einzelner Rechnerknoten weitere Aspekte zu berücksichtigen. Ist weiterhin k kleiner als n, so müssen mehrere Partitionen einem Prozessor zugewiesen werden. Um Verfahren und Kriterien, nach denen eine Partitionierung und das Mapping dieser Partitionen auf einen Parallelrechner erfolgt, soll es in diesem Kapitel gehen. Weiterhin werden Optimierungsansätze diskutiert, die die Laufzeit der parallelen Simulation verbessern sollen, indem applikationsspezifische Informationen, die erst zur Laufzeit einer Simulation feststehen, bei der Objektzuordnung zu einer Partition berücksichtigt werden.

Grundsätzlich haben Modellpartitionierungsverfahren starke Ähnlichkeit mit konventionellen Graphpartitionierungsalgorithmen. Bei den von uns verwendeten Methoden werden Objekte auf Knoten eines Graphen abgebildet. Die Kanäle, die zwei LPs verbinden, stellen die Kanten des Graphen dar. Bei der Simulation von Schaltkreisen haben wir außerdem das Problem, daß die Signale durch Nodes repräsentiert werden, die mehrere Ausgänge mit mehreren Eingängen verbinden

können. Sie stellen somit ein Netz von Kanälen dar, die für die Partitionierung in einzelne Kanäle aufgebrochen werden müssen.

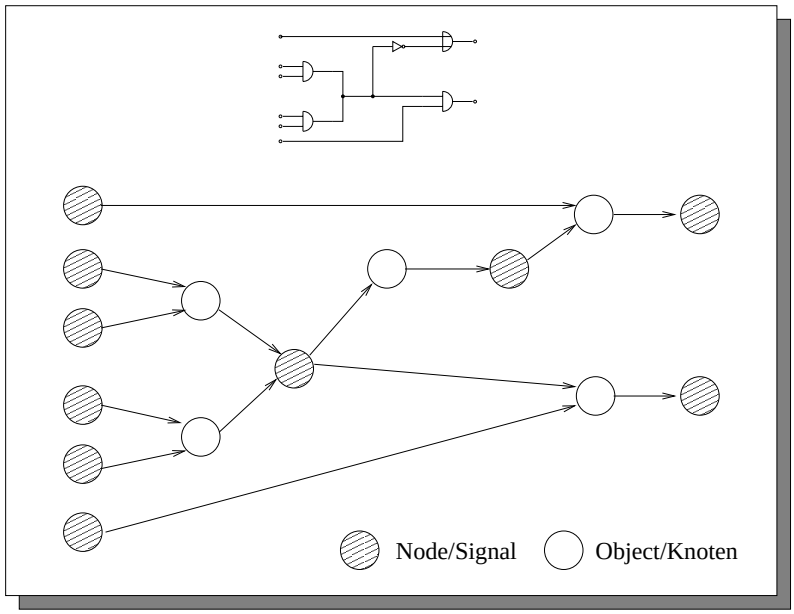


Abbildung 5.1: Abbildung eines Schaltkreises auf einen bipartiten Graphen

Insbesondere ist der Unterschied zwischen Knoten eines allgemeinen Graphen und dem Begriff Knoten bei der Modellierung von Schaltkreisen zu beachten. Die beiden Terme sind hier orthogonal zu betrachten, da ein Node einer Schaltung einer oder mehreren Kanten des allgemeinen Graphen entspricht. Ebenso wird jedes Objekt oder Schaltelement zur Partitionierung auf einen Knoten eines allgemeinen Graphen abgebildet.

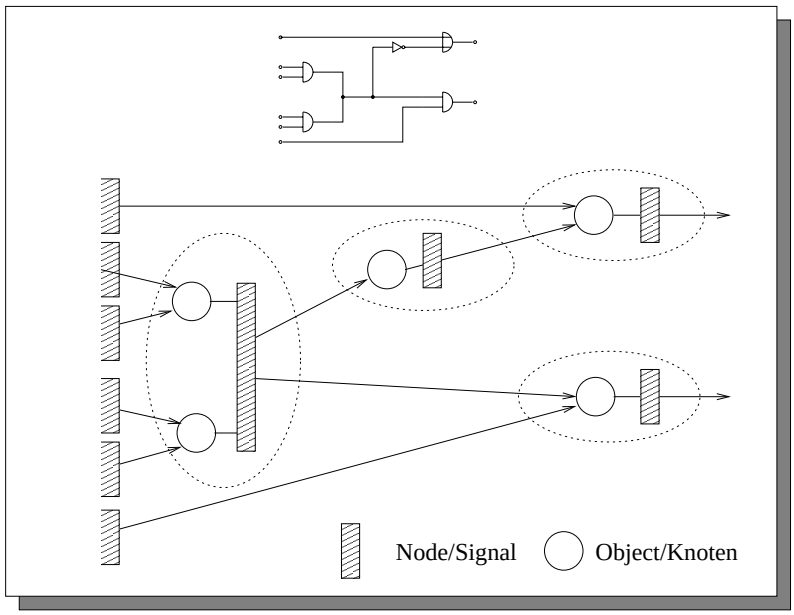


Abbildung 5.2: Abbildung eines Schaltkreises auf einen allgemeinen Graphen

Exakt handelt es sich bei der Abbildung von Schaltkreisen auf allgemeine Graphen um die Modellierung eines bipartiten Graphen, wobei die Nodes und die Objekte die unterschiedlichen Knotentypen repräsentieren und die Verbindungen zwischen Ausgangsobjekt und Node bzw. Node und Eingangsobjekt die Kanten des Graphen darstellen (Abb. 5.1).

Zur Realisierung der Treiberfunktionalität (Entscheidungsfunktion) war bereits im letzten Kapitel gefordert worden, daß alle treibenden Objekte auf dem selben Prozessor plaziert werden. Insbesondere sollten sie auch in der gleichen Partition liegen. Für jedes getriebene Objekt wird eine eigene Kante vom Treiber zum Ziel eingeführt. Durch die Zusammenfassung der Treiber eines Signals zu einem virtuellen Superknoten kann die Abbildung eines Schaltungsmodells jedoch auch als Graph mit nur einem Knotentyp realisiert werden (Abb. 5.2), wodurch die Partitionierungsalgorithmen übersichtlicher werden und sich aufgrund der reduzierten Kanten- und Knotenanzahlen schneller berechnen lassen.

5.1 Partitionierungsalgorithmen

Zur Durchführung von Partitionierung wird im folgenden vorausgesetzt, daß die Modellbeschreibung in Form eines Graphen G =(V, E) mit $|V|$ Knoten (Objekten) und $|E|$ Kanten (aufgesplittete Netze / Nodes) vorliegt. Dieser Graph kann für unterschiedliche Algorithmen gerichtet oder ungerichtet sein. Ziel allgemeiner Partitionierungsverfahren ist die Optimierung verschiedener Aspekte.

1. Es wird davon ausgegangen, daß die Zuteilung eines Knotens für den entsprechenden Prozessor eine gewisse Last generiert. Die Kosten eines Knotens sind in der Regel applikationsspezifisch und müssen dem Algorithmus in Form von Knotengewichten zur Verfügung gestellt werden. Existieren keine solchen Daten, wird idealerweise ein Einheitsgewicht für alle Knoten angenommen. Der Partitionierungsalgorithmus strebt nun ein Lastgleichgewicht zwischen den einzelnen Partitionen an, indem die paarweisen Differenzen der Summen aller Knotengewichte der Prozessoren eine gewisse Grenze nicht überschreiten dürfen. Die Last wird unter der Annahme balanciert, daß die parallele Bearbeitung des Problems am schnellsten erfolgt, wenn alle Prozessoren etwa gleich stark belastet sind.

   Die zugewiesene Last kann durch technologiebezogene Parameter z.B. anhand der Leistungsfähigkeit der verwendeten Prozessoren in heterogenen Umgebungen mittels Gewichten angepaßt werden.

2. Ein weiterer wichtiger Faktor sind die Kosten, die für die Interaktion der Knoten bei der Ausführung eines Algorithmus entstehen. Gewöhnlich werden bei parallelen Berechnungen Daten zwischen den einzelnen an der Berechnung beteiligten Objekten ausgetauscht. Bei der parallelen Simulation handelt es sich dabei um Ereignisnachrichten. In einer verteilten Umgebung kann davon ausgegangen werden, daß der Austausch von Information zwischen Objekten auf verschiedenen Prozessoren höhere Kosten aufgrund Nachrichtenversand oder Synchronisation auf gemeinsamen Speicherbereichen erzeugt als wenn beide auf dem selben Rechner plaziert sind.

   Partitionierungsalgorithmen zielen deshalb neben der Balancierung der Last auf eine Minimierung der Kosten zur Koordinierung der Prozessoren ab. Dazu werden idealerweise ebenfalls applikationsspezifische Daten herangezogen, die den Kanten des Gesamtgraphen Gewichte entsprechend den erwarteten Kosten zuordnen. Liegen keine solchen Informationen vor, so wird ebenfalls mit Einheitsgewichten gearbeitet. Da bei den meisten Architekturen die Kosten

für Nachrichten um ein Vielfaches höher liegen als für eine lokale Synchronisation, können die lokalen Kosten für Objektsynchronisation zur Vereinfachung mit Null veranschlagt werden.

3. Neben diesen beiden Hauptaspekten können auch noch eine Reihe applikationsspezifischer Kriterien bei der Partitionierung eine Rolle spielen. Im Fall der verteilten Schaltkreissimulation zählt dazu z.B. die Forderung nach der Clusterung der Treiber innerhalb einer Partition oder auch die Vermeidung von partitions- bzw. prozessorübergreifenden Zyklen.

Insbesondere die Randbedingungen einer bestimmten Applikationsklasse können zu Zielkonflikten zwischen unterschiedlichen Kriterien führen. Beispielsweise kollidiert die Forderung einer zyklenfreien Partitionierung bei sehr großen Rückkopplungskomponenten meist mit der angestrebten Balancierung.

In die Ermittlung der Gesamtkosten fließen somit verschiedene Größen ein, die unterschiedlich stark gewichtet werden müssen. Ziel eines Partitionierungsalgorithmus ist die Minimierung der daraus resultierenden Gesamtkostenfunktion. Die Verfahren zur Ermittlung einer optimalen Lösung dieses Problems sind bereits für eine reine Graphenpartitionierung mit der Minimierung der Schnittkosten (Anzahl der partitionsübergreifenden Kanten) bei existierenden Verfahren nicht mehr in polynomialer Zeit berechenbar [MEH84a]. Insbesondere bei großen Graphen, die bei der Abbildung von Schaltungen auftreten, lassen sich also nur suboptimale Lösungen mit vertretbarem Rechenzeitaufwand ermitteln. Es kommen insbesondere zur Minimierung von Kommunikationskosten nur Heuristiken zur Lösung des Partitionierungsproblems in Frage.

[JOH96a] betont in einem Übersichtsartikel existierender Algorithmen die Notwendigkeit schneller Partitionierungsverfahren beim Schaltkreisentwurf. Dies gilt einerseits für Floorplaning und Timinganalyse und andererseits auch für Partitionierung zur parallelen Simulation. Die Verfahren können konstruktiven oder iterativen Charakter haben. Bei konstruktiver Vorgehensweise werden die Teilmodelle aus dem Gesamtgraphen schrittweise aufgebaut, indem z.B. starke Zusammenhangskomponenten gebildet werden. Iterative Verfahren versuchen dagegen, bei einer initialen (meist billig zu berechnenden) Aufteilung des Graphen durch Austausch einzelner Komponenten die Lösung bezüglich der Kostenfunktion zu verbessern.

Das Mapping von Partitionen auf spezielle Prozessortopologien wird in der vorliegenden Arbeit nicht betrachtet. Partitionierungen werden in der Regel so generiert, daß für jeden Prozessor genau eine Partition verwendet wird.

Im Prinzip können zum Mapping jedoch rekursiv wieder dieselben Algorithmen eingesetzt werden wie bei der vorangegangenen Partitionierung. Jede Partition wird dabei als ein Knoten betrachtet. Die Kanten zwischen allen Objekten zweier Partitionen faßt man zu einer einzigen Kante zusammen, deren Gewicht sich als Summe der Einzelgewichte ergibt. Für die Kantengewichte kann ein Mappingalgorithmus auch die Topologie des Kommunikationsmediums, auf das der Graph abgebildet wird, z.B. über die Kosten pro Zwischenknoten (Hop) berücksichtigen.

In den folgenden Abschnitten werden einige Partitionierungsverfahren vorgestellt, die bei Schaltkreisaufteilung zum Einsatz kommen. Sie sind prinzipiell aber auch für beliebige andere Applikationen einsetzbar.

5.1.1 Lastbalancierende Verfahren

Einfache Algorithmen, die unter der Annahme, daß alle Objekte während der gesamten Simulation homogenes Lastverhalten zeigen, eine vollständige Balancierung der Partitionsgrößen erreichen, aber keine Optimierungen hinsichtlich des Kommunikationsaufwands unternehmen, sind

- *Natural* [SMU87a]: Das Modell wird anhand seiner Beschreibung in n Blöcke zerlegt. Die ersten $m = \lceil \frac{1}{n} \rceil$ Objekte werden dem ersten Prozessor zugewiesen, die zweiten m Elemente dem zweiten usw. Generiert jedes Objekt eine Einheitslast, so unterscheiden sich die Partitionsgrößen um maximal ein Objekt und können als nahezu ideal balanciert betrachtet werden. Verursachen verschiedene Objekte unterschiedlichen Aufwand bei der Berechnung, so kann die Gesamtlast als Maß für die Partitionsgröße dienen. Die Last der einzelnen Partitionen weicht aber dadurch mitunter stärker voneinander ab, da der exakte Wert m in der Regel nicht genau durch die sukzessive in der Beschreibung angetroffenen Elemente erreicht wird.

  Je nach Art der Erzeugung der Modellbeschreibung können hier lokal innerhalb des Modells stark zusammenhängende Objekte, die eine gewisse Lokalität aufweisen, zu einer Partition zusammengefügt werden oder willkürlich zusammengestückelte Objekte eine Einheit bilden. Als Beispiel für die beiden Extrema ist z.B. die komponentenweise Anordnung mit Auflistung aller Subkomponenten oder die Speicherung nach Objekttyp (Gatter) zu nennen.

  Aufgrund der Abhängigkeit der generierten Partitionen von der Reihenfolge innerhalb der Modellbeschreibung läßt sich somit keine allgemeine Aussage über die zu erwartende Güte der Lösungen machen. Auf jeden Fall ist die Berechnung der Aufteilung sehr billig und in linearer Zeit abhängig von der Modellgröße durchführbar.

- *Random* [SMU87a]: Die Verteilung der Objekte erfolgt durch Ermittlung der Partition über einen im Intervall $[0.5, n+0.5[$ gleichverteilten Zufallszahlengenerator und Rundung auf ganzzahlige Werte. Bei guten Generatoren ist zu erwarten, daß alle Partitionsgrößen innerhalb sehr kleiner Abweichungen gleich groß sind.

- *Round-robin* [MEI93a]: Die Objekte werden einzeln reihum den einzelnen Partitionen zugewiesen. Somit werden das 1., das $n+1$., das $2n+1$. Objekt, usw. der ersten Partition zugewiesen. Analog wird mit den Objekten für die 2-te bis n-te Partition verfahren.

  Für die Bewertung dieser beiden letzten Verfahren gelten ähnliche Gesichtspunkte wie bei Natural.

Unter der Annahme schneller vollvermaschter Rechnernetze mit verschwindend kleinen Kommunikationskosten sollte ein solches lastbalancierendes Verfahren bereits gute Ergebnisse liefern. Weiterhin läßt es sich bei datenparallelen Ansätzen, bei denen die Kommunikation phasenweise nach Erreichen einer Barriere durch alle beteiligten Prozessoren erfolgt, gut einsetzen.

Allerdings trifft man idealisierte Kommunikationsmedien in der Praxis so gut wie nicht an. Switched Networks bieten sofern sie für kollisionsfreie Verschaltung konzipiert sind (Omega, Crossbar, Butterfly) hier die beste Alternative. Sie sind schnell aber auch mit hohen Anschaffungskosten verbunden. Bei Einsatz von Hypercubes, CubeConnectedCycles und 3- bzw. 2-dimensionalen Tori und Gitterstrukturen (Meshs) [HWB84a] müssen Prozesse ggf. über mehrere Hops Nachrichten weiterleiten. Bei einem Ring oder Bus treten zusätzlich verstärkte Lastprobleme auf. In (virtuellen) Shared-memory-Systemen verlagern sich die Probleme wiederum lediglich von der Applikations- auf die Cache- bzw. Synchronisationsebene.

5.1.2 Schnittkostenminimierende Verfahren

Bei heterogenen, verteilten Applikationsstrukturen, deren Komponenten asynchron miteinander kommunizieren sollen, sind die einfachen Verfahren ebenfalls nicht sonderlich geeignet, weil sich der Nachrichtenaufwand nicht oder nur schwer abschätzen läßt. Aus diesem Grund wurden viele

Algorithmen vorgeschlagen, die die Kosten für die Kommunikation zwischen Partitionen, die auf verschiedenen Prozessoren plaziert werden, reduzieren. Diese minimalen Schnittkosten geben einer großen Klasse von Verfahren ihren Namen *Min-cut*. Aufgrund der NP-Vollständigkeit des Optimierungsproblems wird zur Beschleunigung der Berechnung einerseits eine suboptimale Lösung akzeptiert und andererseits Applikationswissen in die Verarbeitung zusätzlich eingebracht. Dazu zählen Informationen über den Koppelungsgrad zwischen Objekten (und daraus abgeleitet die Häufigkeit und Länge möglicher Nachrichten), die Granularität der lokalen Berechnungen und ihr Verhältnis zur Nachrichtenübertragungsdauer oder Wissen über das implizite Verhalten anderer Objekte (z.B. Lookahead).

5.1.2.1 Iterativ optimierende Ansätze

Zur Klasse der optimierenden Algorithmen zählen die Bipartitionierungsverfahren. Ihr bekanntester Vertreter ist der Algorithmus von *Kernighan und Lin (KL)* [KEL70a], der die Aufteilung eines Graphen in zwei nahezu gleich große Teile berechnet. Er startet mit einer durch ein beliebiges Verfahren berechneten (in der Regel billigen) Aufteilung der n Knoten in zwei gleich große Hälften. Zur Optimierung werden Teilmengen von Knoten, X und Y, mit identischer Größe in beiden Hälften ermittelt, deren Austausch einen maximalen Gewinn (d.h. die Reduzierung der Schnittkosten) liefert.

Bei Verfahren zur Minimierung der Schnittkosten zwischen zwei Partitionen A und B eines Graphen G = (V, E) werden lediglich die Kantengewichte von G berücksichtigt. In einer Adjazenzmatrix C werden für jede Kante $(a, b) \in E$ deren Kosten $c_{a,b}$ eingetragen. Existiert zwischen zwei Knoten keine Kante, so hat $c_{a,b}$ den Wert 0. Für ungerichtete Graphen ist diese Matrix spiegelsymmetrisch zur Hauptdiagonale.

Die Kosten, die ein Knoten $a \in A$ zu Knoten in der eigenen Partition erzeugt, ergeben sich als Summe der Kosten aller seiner partitionsinternen Kanten.

$$I_a = \sum_{b \in A} c_{a,b}.$$

Entsprechend generiert er partitionsübergreifend (für alle b aus der zweiten Partition B) die externen Kosten:

$$E_a = \sum_{b \in B} c_{a,b}.$$

Bei Wechsel eines Knotens $a \in A$ in die Partition B verringern sich die Schnittkosten von A um den Wert der Differenz[1] $D_a = E_a - I_a$. Wandert gleichzeitig ein Knoten $b \in B$ in umgekehrter Richtung zu A, so tragen die Kosten der Kanten zwischen a und b immer noch zu den externen Schnittkosten bei und sind von D_a und D_b abzuziehen. Ein bezüglich der Gesamtkostenfunktion (minimale Schnittkosten zwischen je zwei Partitionen) effektiver Gewinn G(a,b) ergibt sich durch den Austausch zweier Knoten also genau dann, wenn

$$G(a, b) = D_a + D_b - c_{a,b} - c_{b,a} > 0$$

gilt.

Die Ermittlung der Teilmengen X und Y läuft nun folgendermaßen ab. Initial wird für jeden Knoten k der Wert D_k berechnet. Danach werden in mehreren Iterationen Teilmengen bestimmt,

[1] Die Differenz kann auch negativ werden. In diesem Fall erhöhen sich die Schnittkosten von A.

deren Austausch die Gesamtkosten reduziert. Im i-ten Schritt wird ein Paar (a_i, b_i) mit $a_i \in A$ und $b_i \in B$ gesucht, deren Wert $G(a_i, b_i)$ maximal ist. Die Knoten dieses Paares werden als bearbeitet markiert, sowie a_i in X und b_i in Y aufgenommen. Außerdem werden die Differenzen der verbleibenden Objekte so aktualisiert als ob die beiden Objekte bereits die Partitionen gewechselt hätten. Die Berechnung des aktuellen Teilmengenpaars bricht ab, wenn keine nichtmarkierten Objekte mehr vorhanden sind, oder keine Paare mit $G(a_i, b_i) > 0$ mehr existieren.

Anschließend vollzieht sich der echte Austausch der Teilmengen zwischen den Partitionen und die Markierungen werden wieder entfernt. Ein Durchlauf oder *Pass* ist somit abgeschlossen. Dieses Verfahren iteriert solange keine nichtleeren Teilmengen und somit weitere Verbesserungen gefunden werden. Die Gesamtkomplexität des Verfahrens wird mit $O(n^2 \log n)$ angegeben.

Die Ermittlung des jeweils optimalen Paars läßt sich durch Verwenden sortierter Listen von Objekten entsprechend fallender Differenzen optimieren, so daß nach Angaben von Kernighan und Lin in der Regel nicht alle Objekte der Listen paarweise miteinander verglichen werden müssen. Weitere Optimierungen zur Ermittlung der Paare mit maximalem Gewinn werden von Dutt und Deng beschrieben [DUT93a, DUD96a].

Aufgrund der initialen Festlegung der Partitionsgröße gestattet der Algorithmus auch eine gewisse Kontrolle der Lastverteilung. Er läßt sich rekursiv zur Generierung mehrerer Partitionen anwenden. Eine ungerade Partitionsanzahl erhält man durch geeignete Wahl der asymmetrischen Splittgrößen bei Zwischenergebnissen.

Alternativ kann die Startaufteilung auch durch k-faches sequentielles Abspalten von $\lceil \frac{n}{k} \rceil$ Objekten und Optimieren entsprechend dem Basisalgorithmus zwischen der jeweiligen neuen Partition und dem nach der Abspaltung verbliebenen Rest ermittelt werden (sequential break-off). Anschließend erfolgt dann eine paarweise Optimierung der gefundenen Partitionen entsprechend obigem Grundalgorithmus, um eine weitere Verbesserung zu erzielen.

Einen etwas modifizierten Ansatz verfolgen *Fiduccia und Mattheyses (FM)*, indem zur Optimierung keine Paare sondern im Wechsel zwischen den Partitionen einzelne Objekte mit maximalem Gewinn verschoben werden [FIM82b]. Die Optimalität der Knotenauswahl leidet dabei unter einer effizienteren Implementierung der Knotensuche, da die Paarauswahl bei Kernighan-Lin (K/L) zusätzlich die Kosten zwischen den zu tauschenden Knoten minimiert. Fiduccia und Mattheyses erreichen bei Annahme von Einheitskosten für die Kantengewichte eine Komplexität von $O(|E|)$. Trifft diese Voraussetzung nicht zu, so läuft der Algorithmus mit $O(n \log n + |E|)$ [DUT93a].

Generell ist zu den Bipartitionsverfahren zu sagen, daß sie nicht generell eine optimale Lösung finden, da aus lokalen Minima kein Ausweg herausführt. Eine Alternative könnte Simulated Annealing sein [KGV83a], dessen Kostenfunktion auch partielle Verschlechterungen gestattet, um insgesamt zu einer besseren Lösung zu gelangen. Johannes äußert jedoch, daß Simulated Annealing keine guten Ergebnisse bei der Bipartitionierung liefert und sehr viel Rechenzeit benötigt [JOH96a].

5.1.2.2 Konstruktiv optimierende Methoden

Die bisherigen Min-cut-Verfahren versuchten eine initiale Partitionierung durch Austausch von Objekten zu verbessern. Im folgenden wird ein Ansatz beschrieben, der gleich von Beginn an versucht, eine im Sinn der Kommunikationskosten gute Lösung schrittweise aufzubauen[2]. Ein solcher Algorithmus, *Soccer* [RUN88a], wurde im SUPRENUM-Projekt (Parallelrechner für numerische Anwendungen) der GMD[3] zur Berechnung der Abbildung von Prozessen auf die verfügbaren freien Rechnerknoten entwickelt.

[2] Die Modifikation von K/L zur Ermittlung der Startaufteilung geht aber bereits in diese Richtung.
[3] Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin

In einem ersten Schritt wird bei der Partitionierung von n Knoten in k Partitionen für jede Partition ein Initialknoten (Centroid) berechnet, um den sich der zugehörige Teilgraph als Cluster bilden kann. Die Auswahl des ersten Knotens ist beliebig. Alle weiteren werden so bestimmt, daß sie von den bereits vorhandenen den maximalen Abstand besitzen. Die Berechnung erfolgt nach einem modifizierten Algorithmus von Dijkstra zur Bestimmung des kürzesten Pfads in einem Graphen.

Ist diese Phase abgeschlossen, wird den gefundenen Partitionen in Round-robin-Weise derjenige der verbliebenen Knoten zugeordnet, der die höchste Summe der Kantengewichte zu dieser Partition aufweist. Als zweites Kriterium wird bei Kostengleichheit die minimale Summe der Gewichte zu anderen Knoten verwendet.

Aufgrund der Reihumverteilung ist eine einigermaßen faire Zuordnung der Knoten zu stark kommunizierenden Zentren zu erwarten. Außerdem läßt sich auch eine gewisse Lastkontrolle in das Verfahren integrieren, indem Prozessoren, deren maximale Partitionsgröße erreicht wurde, übersprungen werden. Die Komplexität des Verfahrens ist durch die Ermittlung der Centroiden mit $O(k \times n^2)$ bestimmt.

5.1.3 Modellorientierte Verfahren

Diese Vorgehensweisen versuchen, die Struktur eines Modells in die Partitionierung mit einzubeziehen. Aus diesem Grund ist es notwendig, nicht mehr beliebige ungerichtete Graphen sondern Graphen mit gerichteten Kanten vorauszusetzen. Durch die Kenntnis der Kommunikationsrichtung lassen sich Eigenschaften der zum Graphen gehörigen Applikation berücksichtigen.

- Die Partitionierung von *Eingangs- bzw. Ausgangskegeln* (Fan-in- / Fan-out-cones) schlagen Smith, Mercer und Underwood vor [SMU87a]. Auf die Schaltkreissimulation bezogen läßt sich durch transitive Clusterung aller Eingangssignale oder aller Ausgangssignal eines Objekts sinnvollerweise die partitionsüberschreitende Kommunikation reduzieren. Außerdem werden dadurch viele Eingangssignale lokal berechnet[4] bzw. nur ein oder wenige Ausgangssignale zum Treiben eines Ausgangskegels benötigt. Die Werte stehen somit eventuell schneller zur Verfügung. Die Berechnung der Cones beginnt idealerweise mit den Primärausgängen bzw. den Primäreingängen eines Modells (im Graph: Knoten ohne ausgehende bzw. eingehende Kanten) und berechnet für jeden Startknoten die Vereinigung aller Objekte, die auf einem Pfad zu einem Primäreingang bzw. -ausgang liegen.

  In der Regel wird es bei diesem Verfahren zu Überlappungen kommen. Nach Ermittlung aller Cones werden die eigentlichen Partitionen bestimmt. Existieren mehr Cones als Partitionen, so werden die Kegel miteinander verschmolzen, so daß danach möglichst große gemeinsame Bereiche verbunden sind. Lastbalancierungskriterien spielen hierbei eine wichtige Rolle.

  Existieren auch nach dieser Zuweisung noch große Überschneidungsbereiche, so kann dies bei einer Simulation zu erheblichem Mehraufwand durch replizierte Bearbeitung führen. Müller-Thuns u.a. sowie Manjikian und Loucks beschreiben diese Vorgehensweise bei Schaltkreissimulatoren [MSD89a, MAL93a]. Sie wählen als Startobjekte der Input-Cones Flipflops, die in der Regel inaktiv sind bis der nächste Clock-Tick erfolgt. Während Müller-Thuns jedoch Replikationen verwendet, blenden Manjikian und Loucks duplizierte Teilkegel auf allen bis auf einer Partition aus. Beide Arbeiten berichten von Partitionierungsdauern unter 30 Sekunden für die großen sequentiellen ISCAS'89-Schaltkreise.

  Überlappungen können aber auch schon vermieden werden, wenn bei der Berechnung der Kegel untersucht wird, ob ein Knoten bereits einmal berücksichtigt wurde und Teil eines anderen

[4] D.h. man muß auf weniger Ereignisnachrichten warten oder weniger verschicken.

Kegels ist. Dadurch werden gemeinsame Teilkegel ausgeblendet und eine schnelle Partitionierung ist in linearer Zeit möglich. Diese Modifikation kommt bei den von uns durchgeführten Untersuchungen zum Einsatz. Eine weitere Limitation zur Vermeidung überproportional großer Cones wurde ebenfalls integriert, indem zu einem Kegel ab einer gewissen Größe keine neuen Objekte hinzugefügt werden. Ein kontrolliertes Wachsen aller parallelen Aus- bzw. Eingänge der Kegel kann durch Verwendung einer rangweisen[5] Breitensuche erreicht werden.

- *String-Partitionierung* [LMP82a] generiert lange Ketten miteinander verbundener Objekte. Insbesondere in Modellen mit Verzweigungen (Fork-Knoten bei Warteschlangennetzen, mehrere Eingänge treibende Signale bei Schaltkreisen) wird davon ausgegangen, daß die auf den Verzweigungen gleichzeitig erzeugten Ereignisse parallel simuliert werden können und deshalb in verschiedenen Partitionen plaziert werden müssen. Die Berechnung erfolgt ebenfalls mit Primärein- oder -ausgängen beginnend als Tiefensuche entlang oder entgegen den Kantenrichtungen. Die verbleibenden Knoten werden entsprechend ihrem Rang als neue Startpunkte weiterer Strings abgearbeitet. Anschließend werden die gefunden Strings reihum auf die verfügbaren Prozessoren verteilt. Eine Längenbegrenzung der Strings sorgt hier ebenfalls für einigermaßen balancierte Submodelle bezüglich der Knotenanzahl.

- Insbesondere bei konservativen Simulationsverfahren bereiten Zyklen und Rückkoppelungen starke Probleme, die sich aufgrund fehlenden Lookaheads in Lawinen von Deadlocks oder Nullnachrichten niederschlagen können. Eine Idee zur Vermeidung solcher Zyklen ist die Zusammenfassung aller Knoten / Objekte einer Schleife innerhalb einer Partition. Der resultierende Supergraph der einzelnen Partitionen ist bei einem solchen Ansatz *azyklisch partitioniert* und enthält anschließend keine Schleifen mehr, die sich über Prozessorgrenzen hinweg erstrecken.

  Beschrieben wird ein solcher Ansatz in einer Arbeit von Sang und Sang [SAS90a]. Kartik und Abraham [KAA92a] weisen jedoch auf die Gefahr durch große Zyklen hin. Gegebenenfalls kann es zu starken Lastungleichgewichten kommen oder die Bildung der gewünschten Partitionsanzahl ist gar nicht möglich. Diese Lastungleichheit durch azyklische Partitionierung läßt sich prinzipiell nicht vermeiden, wenn man Zyklen vollständig eliminieren möchte oder aufgrund des verwendeten Simulationsalgorithmus wenig Alternativen dazu hat.

- Beim *Ranking* werden Objekte entsprechend ihren minimalen Entfernungen zu Primäreinbzw. -ausgängen topologisch sortiert. Objekte mit gleichem Rang werden einer Partition zugeschlagen. Diese Methode findet u.a. bei [KAA92a] und [CCG97a] Anwendung.

5.1.4 Hierarchische Ansätze

Die beschriebenen Basisverfahren lassen sich auch hierarchisch weiter zu komplexeren Methoden zusammensetzen, indem die resultierenden Partitionen wiederum als Graph betrachtet werden, auf den eine weitere Methode angewandt wird. Dies macht jedoch nur dann Sinn, wenn die Anzahl der in der vorherigen Stufe gefundenen Partitionen noch größer ist als die zu benutzende Prozessoranzahl (vgl. Mapping). Außerdem sollte berücksichtigt werden, ob die durch weitere Stufen entstehenden Kosten durch die damit erreichte Verbesserung überhaupt amortisiert werden können. Einige interessante Ansätze werden hier nur kurz genannt.

- Nach einer azyklischen Aufteilung eines Graphen können in einer zweiten Stufe die Kommunikationskosten zwischen den Partitionen als Minimierungskriterium dienen. Aufgrund der

[5] Der Rang eines Knotens entspricht der kürzesten Kantenanzahl, um einen Primäreingang bzw. -ausgang zu erreichen.

bereits wesentlich reduzierten Knotenanzahl läßt sich dazu eine Lösung recht effizient berechnen.

Alternativ dazu läßt sich auch auf einer initialen azyklischen Partitionierung aufsetzen. Anschließend werden für einen Bipartitionsalgorithmus nur Verschiebungen von Objekten zugelassen, die keine neuen Zyklen einführen [CLB94a].

- Bauer und Sporrer schlagen ein Verfahren vor, das zunächst konfluente Schaltungsteile (Verzweigungen, die später wieder bei einem Objekt zusammengeführt werden) extrahiert, die die sogenannten Petals (Blütenblätter) bilden. Anschließend werden diese Blätter als Blütenkränze (Corollas) zusammengruppiert [BAS93b]. Die Autoren berichten von einer verbesserten Leistung bezüglich Partitionierungsgeschwindigkeit und Qualität der Lösung gegenüber anderen Verfahren.

- Karthik und Abraham verwenden gar ein dreistufiges Verfahren [KAA92a]. Zunächst werden Cluster aus starken Zusammenhangskomponenten gebildet, die (im Kontext ihrer Schaltkreissimulation auf Transistorebene) aus direkt miteinander verbundene Transistoren bestehen. Diese relativ kleinen Komponenten werden anschließend zyklenfrei aufgeteilt und topologisch sortiert. Die resultierenden Schaltungsteile werden dann anhand ihres Rangs und der Transistorzahl zu den endgültigen, balancierten Partitionen geformt.

5.1.5 Untersuchungen zum Leistungsverhalten

Smith, Mercer und Underwood untersuchten die Leistungsfähigkeit verschiedener Partitionierungsalgorithmen im Bereich der Schaltkreissimulation [SMU87a]. Die Kegelpartitionierungen schnitten dabei relativ gut ab. Betrachtungen zur Leistungsfähigkeit werden allerdings auf Basis der parallel ausführbaren Ereignisse mit gleichen Zeitstempeln gemacht.

Kravitz und Ackland sprechen denjenigen Verfahren, die Last balancieren oder eine zufällige Verteilung vornehmen, eine bessere Leistung zu als kommunikationsminimierenden Verfahren [KRA88a]. Ihre Untersuchungen basieren im wesentlichen auf einem synchronen Verfahren auf einem Distributed-memory Hypercube mit 64 Prozessorknoten.

Briner verwendete verschiedene Algorithmen zur Partitionierung der untersuchten Schaltungen (Random, hierarchische Strukturierung und Bipartitionierung ohne und mit unterschiedlicher Kantengewichtung) [BRI90a]. Erstaunlicherweise erwies sich in seiner Time Warp-Implementierung mit Fenstertechnik zur Beschränkung des Optimismus die Random-Partitionierung als beste Lösung.

Untersuchungen von Cloutier u.a. [CCG97a] berücksichtigen die Plazierung auf einem heterogenen Workstation-Cluster verschiedener Sun-Rechner. Für kombinatorische und sequentielle Schaltkreise stellt sich die Partitionierung anhand der Ränge (Ordnung von den Eingangs- zu den Ausgangssignalen) für Time Warp als effektiver heraus als eine Minimierung der Kommunikationskosten. Bei der Partitionierung sequentieller Schaltungen, die ein Aufbrechen der Zyklen an den FF-Eingängen durchführt, empfiehlt es sich, die Primäreingänge der Schaltungen den schnelleren Prozessoren zuzuweisen und die Last (Größe der Partitionen) bei den langsameren Rechnern zu reduzieren. Die kombinatorischen Schaltungen weisen gerade bei einer umgekehrten Zuordnung der Eingänge zu langsameren Rechnern ein besseres Verhalten auf, weil dadurch das übereilte Ausführen von Ereignissen in hinteren Schaltungsteilen gebremst wird. Eine Relation zu sequentiellen Simulationsläufen wird nicht angegeben. Lediglich Meßwerte für Simulationen auf 6 Rechnern mit unterschiedlichen Lastparametern werden präsentiert.

[KOT94a] berücksichtigt ebenfalls die Leistung einzelner Arbeitsplatzrechner bei der Partitionierung und versucht weiterhin, die Kommunikation durch einen Min-cut-Algorithmus zu reduzieren.

Gleichzeitig wird Vermeidung von prozessorübergreifenden Zyklen empfohlen.

5.1.6 Parallelisierung von Partitionsalgorithmen

Einen anderen Ansatz, die Laufzeiten von Partitionierungsverfahren zu reduzieren, stellt die Parallelisierung dieser Algorithmen selbst dar.

Lanchès verwendet eine Methode zur parallelen Berechnung von Simulated-annealing auf einem Transputercluster [LAB92a, LAB92b] mit der geschätzten Rechen- und Kommunikationslast der Schaltelemente und Signale als Kostenfunktion.

Nandy und Loucks [NAL93a] versuchen, aufgrund der hohen Kosten bei der auf Heuristiken basierenden Berechnung von Partitionen den Partitionierungsalgorithmus ebenfalls zu parallelisieren. Sie erzielten dabei für einen Min-cut-Algorithmus auf 8 Prozessoren Beschleunigungen zwischen 2,7 und 6,6 für verschiedene kleinere Schaltungen bei gleicher Güte der Ergebnisse im Vergleich zum sequentiellen Partitionierungsalgorithmus. Interessanterweise treten bei der Parallelisierung des Min-cut-Verfahrens ähnliche Probleme wie bei der voll dynamischen Lastbalancierung bezüglich der Auswahl der zur Optimierung zu verschiebenden Gatter auf. Durch einen unkontrollierten und vollständig auf lokalem Wissen einzelner Prozessoren aufbauenden Algorithmus können dabei Instabilitäten auftreten, die die Leistung der Verfahren mindern anstatt sie zu verbessern.

5.2 Adaptivität verschiedener Verfahren

Entsprechend dem Umfang der über das Verhalten einer Applikation zur Laufzeit verwendeten Informationen werden Partitionierungsverfahren im folgenden Abschnitt nach statischen, semidynamischen und dynamischen Ansätzen klassifiziert [KRA88a].

5.2.1 Statische Verfahren

Die beschriebenen Vorgehensweisen setzen voraus, daß sie ein ausreichendes Wissen über das Verhalten der Modelle während der gesamten Laufzeit einer Applikation besitzen. Insbesondere zählen dazu die Last, die die mit einzelnen Knoten assoziierten Berechnungen generieren, und das Kommunikationsaufkommen. Die Festlegung der erwarteten Kosten erfolgt vor Beginn der Partitionierung und bleibt in der Regel für eine Vielzahl von Modellen einer Applikationsklasse fest. Beispielsweise wird ein konstantes Nachrichtenaufkommen als Gewicht für alle Kanten und eine vorgegebene mittlere Granularität der Bearbeitungsschritte für alle Knoten vorausgesetzt. Die beschriebenen Algorithmen sind deshalb als hochgradig *statisch* zu bezeichnen. Stellt sich während der Bearbeitung heraus, daß die Annahmen unzutreffend sind, so müßte die Partitionierung mit den neuen Daten wiederholt und die Applikation neu gestartet werden.

Weist die Anwendung gar unterschiedliche Phasen auf, die eigentlich verschiedene Partitionierungen erfordern, so läßt sich das erst nach Abschluß der Berechnung erkennen. Eine Lösung dieses Problems kann durch eine Approximation der optimalen Parameter versucht werden, was aber wiederum zusätzlichen Berechnungsaufwand durch Optimierung und Wiederholungen der Simulationsläufe generiert. Die Güte der Ergebnisse ist außerdem von der Stärke der Schwankungen abhängig und oft nicht homogen für alle bearbeiten Modelle einer Applikationsklasse.

5.2.2 Semidynamische Vorgehensweise

Als Möglichkeit, detailliertere Informationen über das Verhalten eines konkreten Modells zu erlangen, wird von verschiedenen Autoren vorgeschlagen, das Modell für eine kurze Zeit (oder komplett)

ablaufen zu lassen und dabei Daten über das konkrete Verhalten der Applikation zu ermitteln (*Presimulation*, [MAL93a]). Die Knotengewichte lassen sich dann anhand der tatsächlich beobachteten Last festlegen. Die Kantengewichte bestimmen sich aus dem realen Kommunikationsaufkommen. Mit den so ermittelten Parametern läßt sich eine vermutlich verbesserte Partitionierung berechnen.

Problematisch ist allerdings hier ebenfalls der zusätzliche Berechnungsaufwand zur Bestimmung der realen Parameter. Je nach Problemgröße kann das Ermitteln der Daten erhebliche Kosten durch die Presimulation erzeugen. Der zusätzliche Aufwand besteht dabei in der Laufzeit eines sequentiellen oder verteilten Simulators, der die Daten generiert. Der eigentliche Partitionierungsanteil an den gesamten Simulationsnebenkosten erhöht sich dadurch nicht, da für die Berechnung der gemessenen Parameter ein sequentieller Simulator oder ein verteilter Simulator verwendet werden kann, der mit einer extrem billigen Primitivpartitionierung startet. Durch die Laufzeit der Presimulation selbst ergibt sich jedoch insgesamt eine erheblich längere Laufzeit.

Dynamische Schwankungen während der Laufzeit werden allerdings beim Ansimulieren nicht berücksichtigt. Ändert sich das Verhalten im Anschluß an den simulierten Zeitraum drastisch, so stellt sich die Frage nach der Repräsentativität der gefundenen Parameter für das tatsächliche Verhalten. Selbst bei einem kompletten Durchlauf stellen die Werte nur gemittelte Größen dar. Schwanken sie im realen Ablauf stark zwischen zwei Extrema, so ergeben sich daraus wieder Leistungsverluste. Chamberlain und Henderson ermittelten bei Untersuchungen der Stabilität des Verhaltens von 3 Schaltkreisen (mit bis zu 3000 Objekten) eine große Übereinstimmung der in den ersten 10 Prozent einer Simulation ermittelten Parameter mit der restlichen Simulationszeit [CHH94a]. Dabei wurden 100 verschiedene Stimulidatensätze verwendet, so daß eine zufällige Übereinstimmung weitgehend ausgeschlossen werden kann.

Manjikian und Loucks setzten Presimulationsläufe zur optimierten Berechnung realistischerer Schnittkosten bei der Verteilung ihrer Input-Cones auf Partitionen ein. Die Länge der Vorsimulation betrug 4 Prozent der Gesamtlaufzeit. Im Gegensatz zu [CHH94a] wurden Lastungleichgewichte bei den Ereigniszahlen der einzelnen Knoten beobachtet, die bei Verwendung von Statistiken aus einem vollen Simulationslauf weitgehend vermieden wurden.

Kim und Jean bestimmten die Aktivitäts- und Kommunikationsraten durch Presimulation für Unit-delay-Logiksimulationen auf Gatterebene [KIJ96a]. Zunächst wird eine rangweise Sortierung des Graphen beginnend mit den Primäreingängen durchgeführt. Jeder Primäreingang wird einer Partition zugewiesen. Jeder weitere Knoten wird derjenigen Partition zugeschlagen, die entsprechend den ermittelten Werten die höchsten Kosten zu ihm aufweist. Diese Partitionen werden intern anschließend entsprechend ihren Kosten nochmals sortiert und nur benachbarte Partitionen einem Prozessor zugewiesen. Die Partitionierung großer sequentieller ISCAS'89-Benchmarks [BBK89a] dauerte weniger als 5 Sekunden. Die Messungen erfolgten mit 100 zufälligen Eingabevektoren. Für die Presimulation wurde $\frac{1}{10}$ der Gesamtsimulationszeit berücksichtigt. Speedups von 31 auf 64 Knoten wurden gegen einen sequentiellen Simulator bei Verwendung eines großen Schaltkreises (s38584.1) gemessen.

Cloutier u.a. [CCG97a] beobachteten bei Verwendung der Ergebnisse aus Presimulationsläufen zur Partitionierung von Schaltkreissimulationen eine Geschwindigkeitssteigerung um 50 Prozent gegenüber dem Basisverfahren des Time Warp.

5.2.3 Dynamische Methoden

Leider generieren alle bisher beschriebenen Verfahren einen nicht unerheblichen zusätzlichen Aufwand, um überhaupt erst einmal eine parallele Simulation starten zu können. Idealerweise sollte eine parallele Applikation in der Lage sein, mit einer beliebigen (billigen) Aufteilung zu starten und

durch Beobachtung und automatische Migration geeigneter Teilkomponenten einen konvergenten stabilen Zustand mit ausgeglichenen Belastungen zu erreichen. Ändert sich die Lastsituation, so soll ein solches Verfahren erneut starten und eine weitere Adaption vornehmen.

Die Beantwortung einer Reihe von Detailfragen wird in Kapitel 9 thematisiert. Hier werden lediglich einige Herangehensweisen an das nichttriviale Problem der automatischen Lastbalancierung beschrieben.

Eine allgemeine Einführung in Methoden der Lastbalancierung geben Shivaratri u.a. in [SKS92a]. In [KGR94a] wird die Frage der Skalierbarkeit eines dermaßen komplexen Ansatzes diskutiert und ein Maß für die Wachstumsrate eines Problems definiert, um bei steigender Problemgröße in gleichbleibender Zeit auf einem Parallelrechner mit linear wachsender Prozessoranzahl gelöst zu werden. In [SKP93a] erörtern Shanker u.a. einen theoretischen Ansatz, die Ankunftsrate von Jobs vorherzusagen, um eine rechtzeitige geeignete Umpartitionierung zur Laufzeit vornehmen zu können. Konkretere Ansätze und umgesetzte Algorithmen finden sich in folgenden Arbeiten.

- Reiher und Jefferson schlugen in einem frühen Ansatz das zeitliche Aufteilen logischer Prozesse im TWOS vor. Basis ihrer Lastberechnung war die effektive Arbeit, die ein Prozeß leistet (*effective utilization*). Bei diesem Maß wird nur der Rechenzeitanteil derjenigen simulierten Ereignisse, die nicht zurückgesetzt werden, berücksichtigt [REJ90a].

- Simic und Ortner fanden die dynamische Repartitionierung von Schaltungen nur bei kleiner Partitionsanzahl hilfreich [SIO93a]. Sie betrachteten einen Mixed-Level Schaltkreissimulator unter Time Warp mit einer zentralen Steuerung der Umverteilung im laufenden Betrieb. Es wurden nur leichte Verbesserungen bezüglich der Leistung des Time Warp beobachtet. Generell lagen die Speedups unter 2 bei bis zu 5 Prozessoren.

- Luksch [LUK93a] beschreibt die Anforderungen an dynamische Lastbalancierungskomponenten und schlägt als Lastmaß die Differenz der LVT und GVT vor. Konkrete Ergebnisse aus Untersuchungen liegen nicht vor.

- Schlagenhaft, Ruhwandl und Sporrer entwickelten im Rahmen eines optimistischen Schaltkreissimulators [BAS93a] auf Basis eines Workstationnetzes eine dynamische Lastbalancierungskomponente [SRS95a]. Ergebnisse der Leistungsfähigkeit liegen bis auf ein 2-Prozessor-Modell nicht vor. Als Austauschkomponenten werden komplette Partitionen verwendet, die vor Simulationsbeginn mit einem statischen Algorithmus berechnet wurden. Dadurch sollen Kosten bei der Auswahl zu verschiebender Objekte reduziert werden. Als Lastmaß wird das relative Fortschreiten der lokalen Simulationszeit skaliert über die Realzeit verwendet. Da die GVT von einem zentralen Prozeß berechnet wird, fungiert dieser auch als Kontrollorgan für die Lastbalancierung.

- Avril und Tropper [AVT96a] beschreiben ein ähnliches Verfahren wie bei Schlagenhaft, Ruhwandl und Sporrer ebenfalls für Time Warp und setzen auch auf einer initialen Partitionierung (mit Strings) auf. Auch hier werden nur ganze Cluster zwischen Knoten eines BBN GP1000-Rechners verschoben und Einsparungen zwischen 20 und 50 Prozent verglichen mit der Laufzeit des Basis-Time Warps für die beiden größten ISCAS'89-Benchmarks angegeben. Als Lastmaß wird der sogenannte *Throughput* eines Simulators verwendet, der vergleichbar mit Reihers Utilization ist.

- Boukerche und Das beschreiben die einzige sonst bekannte Realisierung eines Lastbalancierungsverfahren für konservative Simulationsprotokolle (mit Nullnachrichten) [BOD97a].

Es werden unter zentraler Kontrolle stets mehrere zusammenhängende Objekte verschoben, die dynamisch entsprechend ihres Kommunikationsaufwands untereinander bestimmt werden. Als Lastmaß dient die Anzahl der aktuell gespeicherten, unbearbeiteten Ereignisse und Nullnachrichten. Auf Intel Paragon- und iPSC/860-Rechnern werden Speedups von 5 – 6 auf 16 Prozessoren mit Modellen der Größenordnung 300 – 500 Objekte erreicht. Der Referenzwert (sequentieller Simulator oder paralleler Simulator auf einem Knoten) ist jedoch aus der Beschreibung nicht ersichtlich.

- Der Ansatz von Wilson und Nicol [WIN96a] ist trotz seines Titels eher den semidynamischen Varianten zuzuordnen, die durch Presimulation die Leistung statischer Partitionierungsverfahren verbessern sollen.

5.2.4 Resümee

Neben den grundlegenden statischen Verfahren zur Lastbalancierung wurden auch semidynamische Ansätze in unsere Evaluierungsumgebung integriert. Nach einer Simulationsstartphase kann der Schaltkreis automatisch mit den ermittelten Daten wieder eingesammelt, neu partitioniert und entsprechend der geänderten Zuordnung wieder verteilt werden.

Diese Ansätze stellen eine Vergleichsmöglichkeit zu bisherigen Arbeiten bereit, um die vielfältig gemachten Beobachtungen besser einordnen zu können. Einen weiteren Aspekt der Arbeit stellt die Untersuchung dynamischer Lastbalancierungsverfahren dar, die den Benutzer vollständig vom zusätzlichen Aufwand der Partitionierung vor Simulationsbeginn befreien. Im Rahmen dieser Dissertation wurde ein dezentraler Algorithmus ohne explizite Kontrollkomponente entwickelt, der transparent für den Simulator eine Verschiebung von Objekten zur Laufzeit ermöglicht und so eine automatische Ausbalancierung der Last zu Laufzeit gestatten soll. Das Verfahren ist sowohl unter optimistischen als auch konservativen Algorithmen einsetzbar. Inwieweit es die in es gesetzten Erwartungen erfüllen kann, wird in Kapitel 9 diskutiert.

Teil II

Analysen, Interpretationen und Ergebnisse zum Leistungspotential paralleler Simulationen — Wann lohnt sich Parallelisierung?

Kapitel 6

DVSIM – Eine Umgebung zur Evaluierung paralleler Simulation

Nachdem im ersten Teil sehr umfassend existierende Ansätze zur parallelen Simulation beschrieben, die Verfahren detailiert dargelegt und verschiedenste Einflußfaktoren wie Partitionierung, Synchronisationsverfahren und Modellierungsaspekte besprochen wurden, sollen diese Bruchstücke nun im zweiten Teil wieder zu einem Gesamtbild zusammengesetzt werden, das als breite und umfassende Darstellung die Frage beantwortet, ob sich die Parallelisierung diskreter ereignisgesteuerter Simulation im Sinne erreichbarer Beschleunigung lohnt und unter welchen Bedingungen eine solche Vorgehensweise sinnvoll ist.

Dazu wurde eine Reihe von Simulatoren für ein Framework entwickelt, deren Kern der parallele Simulator DVSIM zur Schaltkreissimulation als exemplarische Anwendungsklasse darstellt. Daneben besteht die Evaluierungsumgebung aus einem optimierten sequentiellen Simulator (VSIM), der die Basis der Leistungsmessungen bildet, und Erweiterungen von VSIM. Die Erste dient zur Berechnung des kritischen Pfads für die real zu erwartende Berechnungsdauer eines Simulationslaufs. Die zweite Ergänzung generiert die Protokolldateien für das Oracle-log-Verfahren, mit dem die Leistungsfähigkeit konservativer Simulationsverfahren abgeschätzt werden kann.

Mit Hilfe dieser Komponenten werden reale und künstliche Simulationsmodelle auf ihre grundsätzliche Parallelisierbarkeit untersucht, indem die in Teil I genannten Einflußfaktoren durch Parametrisierung der in DVSIM integrierten parallelen Algorithmen so konfiguriert werden, daß sie ein breites Spektrum der von verschiedenen Forschern vorgeschlagenen Optionen abdecken. Aus der Fülle der ermittelten Informationen werden schließlich die wesentlichen Aspekte extrahiert, um die Ausgangsfrage nach dem Beschleunigungspotential paralleler Simulation umfassend zu beantworten.

Zu Beginn der Überlegungen zum Thema dieser Arbeit stand der Gedanke, die Welt der parallelen ereignisgesteuerten diskreten Simulation in einer strukturierten Einteilung nach wichtigen Parametern sorgfältig zu untersuchen, um ihren Einfluß auf die Gesamtleistung zu beurteilen. Es ergab sich im Laufe der Bearbeitung die Einsicht, daß eine klare Trennung der verschieden Einflußgrößen adhoc nicht möglich ist. Insbesondere übernahmen Aspekte, deren Bedeutung zunächst unterschätzt wurde, eine wichtige Rolle. Dazu zählen insbesondere die Partitionierung und Lastbalancierung der Modellbeschreibung. Neben den grundlegenden Parametern Synchronisationsverfahren, Kommunikationsprinzipien und -medien sowie Granularität nehmen sie aufgrund der Erkenntnis ihrer Bedeutung eine führende Rolle in den weiteren Untersuchungen ein. Beginnend mit den Ausgangswerten der sequentiellen Simulation und den künstlichen Bewertungen durch Kritische-Pfad-Analyse werden zunächst die Partitionierungsverfahren in die Beurteilung mit VSIM und DVSIM

integriert. Danach werden die jeweils besten Ergebnisse als Ausgangspunkt für die Untersuchungen weitergehender Verbesserungsmöglichkeiten entsprechend in der Literatur vorgeschlagener Verfahren verwendet. Weiterhin erfolgt eine faire Berücksichtigung aller entstehenden Kosten, um die Beschleunigungswerte zu ermitteln.

6.1 Grundlagen

Die Entwicklung von DVSIM (**D**istributed **V**HDL **SIM**ulator)[1] [MEI96a] erfolgte auf Basis des sequentiellen Schaltkreissimulators VSIM, der an der University of Pittsburgh von Martello, Levitan u.a. entwickelt wurde [LEV93a]. Dadurch vermieden wir zum einen die aufwendige Implementierung des eigentlichen Simulationskerns und zum anderen stand ein sequentieller Simulator als Vergleichsbasis für die Leistungsbewertung der parallelen Verfahren bereit.

6.1.1 Verfügbarer VHDL-Sprachumfang

VSIM implementiert nicht den vollen VHDL-Standard IEEE 1076 von 1987 [VHD87a], aber weist die wesentlichen Konzepte, eingeschränkt auf eine 3-wertige Logik (0, 1, unknown (u)), auf. Strukturelle Modellierung und Verhaltensbeschreibungen sind ebenso möglich wie die Nutzung des Entity-Architecture-Bindings. Fehlende Komponenten dienen im wesentlichen der leichteren Programmierbarkeit, die aber nur bei der Verwendung innerhalb einer kompletten CAD-Umgebung wichtig sind. Dazu zählen Packages zur strukturierten Erzeugung von Bausteinbibliotheken, Bibliotheken selbst, oder Textein-/ und -ausgabemöglichkeiten, die in der Sprachdefinition vorgesehen sind. Außerdem wurden zunächst nur einfache Datentypen integriert und auf Prozeduren und Funktionen verzichtet.

Andere fehlende Eigenschaften sind Attribute von Signalen und Datentypen, mit denen sich besondere Eigenschaften oder Zustände modellieren lassen[2], Resolutionsfunktionen und dynamische Parametrisierung generischer Bauteile bei der Instantiierung. Diese Mängel lassen sich jedoch, wenn sie unbedingt benötigt werden, teilweise durch Modellierung über zusätzliche VHDL-Prozesse oder ggf. unter Einbindung nativen C-Codes emulieren. Es ist außerdem zu bedenken, daß es sich hierbei um kein kommerzielles Produkt handelt. Der Source-Code kommerzieller Simulatoren war trotz mehrerer Anfragen aus verständlichen Gründen (Bedeutung des Marktwerts einer vollen VHDL-Implementierung) nicht erhältlich. VSIM wird im wesentlichen im universitären Umfeld eingesetzt und stellt eine gute Alternative für unsere Forschungszwecke dar.

6.1.2 Vorverarbeitungsschritte für Simulationsläufe

Die Schaltung wird mit einem konventionellen Editor in VHDL programmiert. Anschließend wird der Code (mit dem Tool `vcomp`) syntaktisch überprüft und in ein Zwischenformat kompiliert, das vom Simulator verwendet werden kann. Dazu wird ein Intermediate-VHDL-Format (IVF) verwendet, das sowohl hierarchische als auch eine "eingeebnete", flache Schaltungsbeschreibungen enthält. Die 1992 erworbene und zugrundeliegende Simulationssoftware liest jedoch nur die flachen Beschreibungen der Schaltung ein und operiert auf der Gatterebene.

Für jeden VHDL-Prozeß wird außerdem eine C-Funktion, die das dynamische Verhalten des Prozesses nachbildet, erzeugt. Besitzt ein Schaltungsmodell keine VHDL-Prozesse, so kann das IVF-File

[1] Der Name des Simulators stellt gewissermaßen noch eine "Altlast" aus den Anfangszeiten des parallelen Simulators dar [SIM94a, WEI94a]. Nach der Einführung des Namens in [MEI93a] wurde von einer Umbenennung z.B. in PVSIM (für **P**arallel ...) abgesehen.

[2] Die Attribute zur Anzeige der Richtung des letzten Signalwechsels - rising oder falling - sind jedoch vorhanden.

direkt von einem generischen Simulator eingelesen werden. Andernfalls muß ein modellspezifischer Simulator aus den zusätzlichen Funktionen und einem generischen Anteil erzeugt werden (Abb. 6.1).

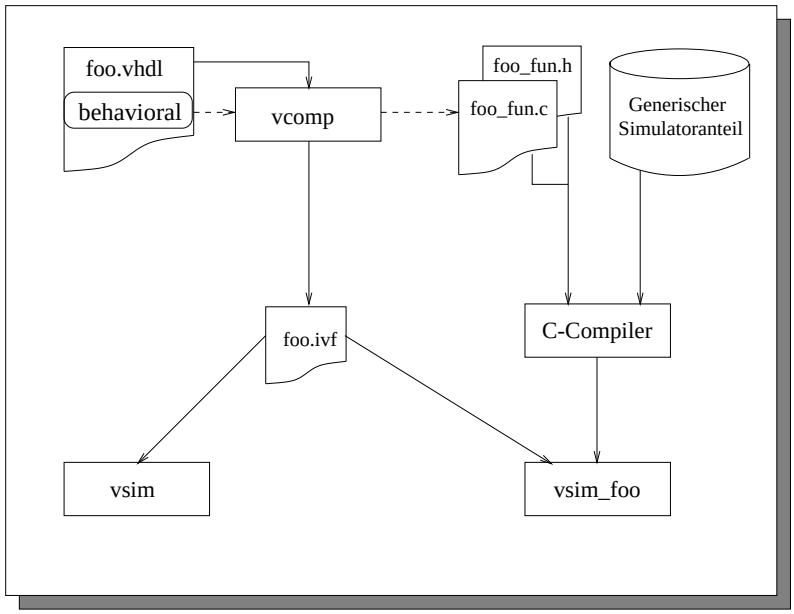


Abbildung 6.1: Entwicklungszyklus eines VHDL-Modells mit (D)VSIM

Diese Vorgehensweise wurde im parallelen Simulator analog realisiert. Auch hier steht ein generischer Simulationskern zur Verfügung, der bei Bedarf mit zusätzlichen Prozeß-Funktionen zu einem modellspezifischen Simulator gebunden werden kann.

6.2 Die Komponenten der Evaluierungsumgebung

Um eine Bewertung von Simulationsverfahren durchführen zu können, sind verschiedene Analysewerkzeuge notwendig. Den Ausgangspunkt der Parallelisierung stellt der sequentielle Simulator VSIM dar. In ihn wurde zunächst ein Analysewerkzeug zur Bestimmung des kritischen Ausführungspfads für beliebige Modelle und weiterhin eine Protokollierungskomponente für kausal abhängige Ereignisse integriert. Alle drei Teile sollen eine Bewertung des Parallelisierungspotentials paralleler Verfahren gestatten bzw. zur Unterstützung der Leistungsanalyse dienen. Die verschiedenen parallelen Protokolle werden im Anschluß an die Beschreibung der sequentiellen Komponenten erläutert.

6.2.1 VSIM – die sequentielle Basis

VSIM stellt dem Benutzer sowohl eine interaktive als auch eine Batchschnittstelle zur Bedienung bereit. Eine Reihe von Kommandos gestattet die Definition von Stimulivektoren für unterschiedliche Signale und spezielle Taktsignale (Clocks) sowie die Festlegung von Beobachtungspunkten für den Bediener interessierende Signale. Weiterhin lassen sich Simulationsläufe über die Befehlsschnittstelle sowohl im Einzelschritt als auch als Ganzes starten. Die Simulation eines VHDL-Modells erfolgt nach den in Kapitel 4 beschriebenen Verfahren.

Die Granularität der Ereignisse kann über eine Umgebungsvariable gesteuert werden. Ist ihr Wert ungleich Null, wird zu Simulationsbeginn eine interne Größe berechnet und so initialisiert,

daß im Anschluß an die eigentliche Ereignisauswertung eine Leerschleife eine verlängerte Berechnungsdauer der Routine simuliert. Ist die Variable unbelegt, so entspricht die Abarbeitung der ursprünglichen Version von VSIM.

6.2.2 Eine CPA-Erweiterung für den sequentiellen Simulator

Durch Aktivierung der Kompileroption CPA werden für jede Ereignisausführung die benötigten Zeitmessungen (Ermittlung der Ereignisbearbeitungsdauer) automatisch ausgeführt und die internen Datenstrukturen um die in Kapitel 3.3.2 definierten Variablen zur Speicherung der frühsten Auswertungszeit erweitert. Da der kritische Pfad stets von einer gegebenen Partitionierung abhängt, liest der Simulator direkt nach dem Start eine als Parameter übergebene Datei ein, in der die Zuordnung der Objekte zu den verwendeten Prozessoren vorgenommen wird. Die Berechnung der frühsten Ausführungszeitpunkte jedes Ereignisses erfolgt eingebettet in die Ausführung der Ereignisroutinen mit sehr geringem Aufwand (wenige einfache arithmetische Ausdrücke und zwei Zeitmessungen). Somit erübrigt sich eine nachträgliche Auswertung und die Ergebnisse liegen unmittelbar nach Ende eines sequentiellen Simulationslaufs vor.

Wird keine Partitionierung angegeben, geht der Simulator von einem Parallelrechner mit unbeschränkter Knotenanzahl aus und berechnet den kritischen Pfad unter der Hypothese, daß jedes Objekt über einen eigenen Prozessor verfügt. Die Kommunikationszeit wird zunächst nicht berücksichtigt, so daß ein Ereignis für seine Folgeereignisse den Endzeitpunkt seiner eigenen Bearbeitung als Generierungszeit einträgt. Die Kombination mit der Einstellmöglichkeit der Ereignisgranularität ist möglich. Die Abfrage der Berechnungsergebnisse des kritischen Ausführungspfads wurde als neues Kommando in die interaktive Bedienschnittstelle integriert.

6.2.3 Protokollierungsmodul für Ereignisaufzeichnung

Um die realen Kosten konservativer Synchronisationsprotokolle abschätzen zu können, wurde eine parallele Implementierung des Oracle-log-Protokolls in DVSIM realisiert. Die Generierung der Daten, die die Ereignisabhängigkeiten widerspiegeln, erfolgt jedoch durch einen sequentiellen Simulator. Alle Ereignisse werden in einer Log-Datei in der korrekten kausalen Reihenfolge ihrer Bearbeitung gespeichert. Die Zuordnung von Ereignissen zu Prozessoren hat ebenfalls auf Basis einer vorgegebenen Partitionierung zu erfolgen. Die Partitionierungsdatei wird ebenso wie bei der CPA-Ermittlung zu Simulationsbeginn eingelesen. Bei der Abarbeitung jedes Ereignisses schreibt der Simulator alle über die Prozessorgrenzen gehenden Signalwechsel in die jeweiligen Log-Dateien aller Zielprozessoren.

Prinzipiell ist die Erzeugung dieser Daten auch mit einem konservativen oder optimistischen parallelen Simulator möglich. Dabei protokolliert jeder Prozessor die von extern kommenden Ereignisnachrichten in eine Log-Datei. Dieser Ansatz wurde jedoch nicht realisiert. Statt dessen verwenden wir den modifizierten sequentiellen Simulator, der um ein Kommando zur Erzeugung der Dateien erweitert wurde.

Da mitunter sehr große Datenmengen generiert werden, existieren alternativ ein kompaktes binäres Datenformat (PLAIN) und für Analysezwecke durch den Benutzer auch eine Textrepräsentation (VERBOSE). Die Kennzeichnung des Typs erfolgt durch ein Flag am Dateianfang.

6.3 Aufbau von DVSIM

Im folgenden werden die einzelnen Komponenten des Simulators und ihre Interaktionsmöglichkeiten kurz beschrieben, sowie Begründungen für gewählte Designentscheidungen gegeben.

6.3.1 Prozeßstruktur

Der parallele Simulator DVSIM lehnt sich bezüglich seiner Schnittstelle für den Benutzer stark an die seiner sequentiellen Ausgangsvariante an. Die Schnittstelle wird innerhalb eines speziellen Prozesses, der Central-Task[3], realisiert.

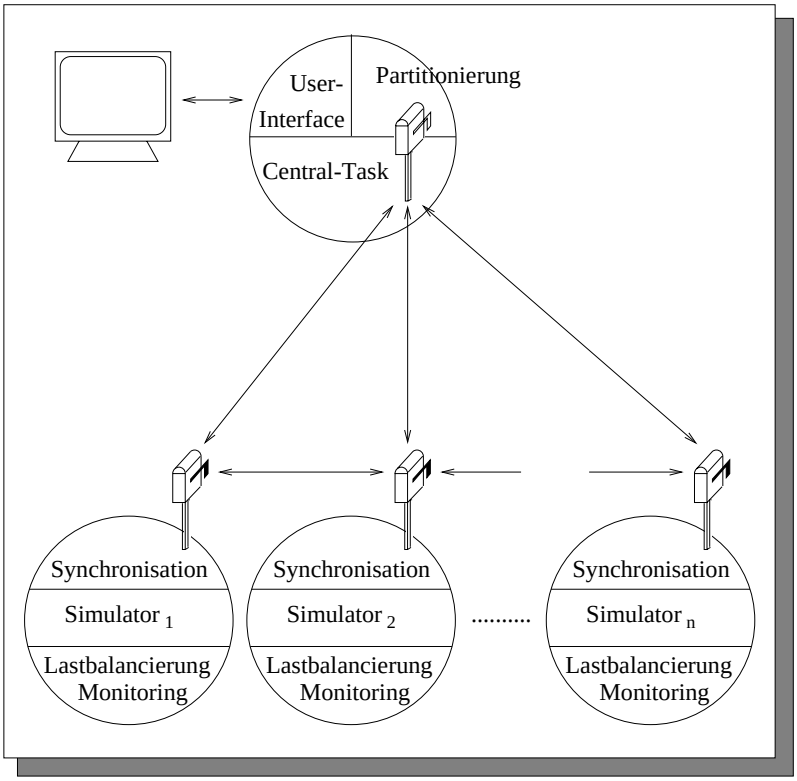


Abbildung 6.2: Aufbau der Komponenten von DVSIM

Bei diesem Prozeß handelt es sich im Prinzip um einen um zusätzliche Kommandos erweiterten, sequentiellen Simulatorrumpf, der nur noch die Rahmenkomponenten von VSIM enthält, aber keine Simulationsfunktionalität mehr beinhaltet. Stattdessen verfügt die Central-Task über Module, die

- verschiedene Partitionierungen für einen eingelesenen Schaltkreises berechnen,
- die gewünschte Anzahl benötigter Simulatorprozesse (Simulator-Tasks) parallel starten,
- die Partitionen an die Simulator-Tasks verteilen,
- einen gemeinsamen Startschuß nach der Verteilung von Schaltung, Stimuli und Simulationssteuerinformation geben,
- die Terminierung der Simulation koordinieren und

[3]Der Begriff Task leitet sich von der ursprünglichen Entwicklungsumgebung MMK (Multiprocessor Multitasking Kernel) [BEL90b], die an der TU München entwickelt wurde, her. In MMK können Tasks über Mailboxen kommunizieren, um somit parallele Applikationen zu realisieren. MMK lief auf Intel iPSC/2 und iPSC/860 Hypercubes sowie auf Sun-Workstations. Aufgrund der eingeschränkten Portierbarkeit wurde DVSIM jedoch recht früh auf den sich abzeichnenden Pseudostandard *Parallel-virtual-machine (PVM)* [SUN90a, SGD93a] umgestellt.

- Ergebnisse einsammeln und ausgeben.

Central- und Simulator-Task werden innerhalb eines einzigen ausführbaren Programms gebündelt. Je nach Aufrufsemantik[4] verzweigen einzelne Prozesse in entsprechend unterschiedliche Funktionalität.

6.3.2 Kapselung der Synchronisationsverfahren

Bei der Realisierung verschiedener Synchronisationsprotokolle wurde versucht, sich an das von Reynolds vorgeschlagene Filterkonzept zu halten [RED89a]. Dazu wurden vier verschiedene Schnittstellen als Interface zur Protokollkapselung definiert. Die unterschiedlichen Module, die diese Interface entsprechend des zu realisierenden Synchronisationsverfahren umsetzen, werden in einer einzigen Datei definiert, die dann zum Simulationskern hinzugebunden wird. Für jedes Synchronisationsprotokoll wird somit (ähnlich wie bei Verwendung der VHDL-Prozesse) ein eigenes ausführbares Programm erzeugt, um die Größe des Codes einigermaßen zu begrenzen. Die Kombination von Central- und Simulator-Task stellt bereits einen Kompromiß dar, der aber eingegangen wurde, weil es sich im wesentlichen um die selbe Komplexität handelt, die ein sequentieller Simulator ebenfalls aufweist.

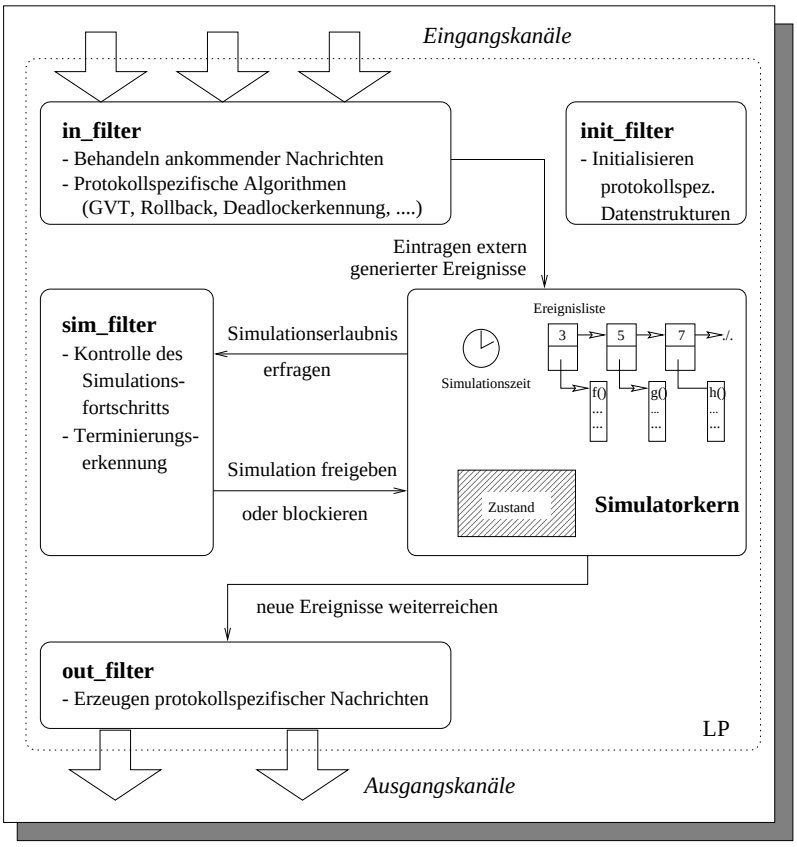


Abbildung 6.3: Filterkonzept in DVSIM

Allerdings ergaben sich ähnlich wie bei Reynolds Probleme mit der vollständigen Kapselung der Protokolle [RWF89a]. Insbesondere stellte sich heraus, daß die Realisierung einer universell

[4]Die von der Central-Task gestarteten Simulator-Tasks erhalten weniger Parameter als die Central-Task selbst.

einsetzbaren Ereignisliste nicht unbedingt für alle Synchronisationsverfahren geeignet ist. Deshalb wurde die Schnittstelle hier um eigene Implementierungen für konservative und optimistische Ereignisverwaltung (mit gleicher Schnittstelle für die Einfüge- und Löschoperationen) erweitert. Im Rahmen dieser Erweiterung konnte auch die Kapselung der Zustandssicherung in die spezialisierte Ereignislistenverwaltung für optimistische Verfahren integriert werden. Es kommt dabei eine mit der Ereignisliste verbundene inkrementelle Checkpointing-Variante zum Einsatz [BAS93a].

6.3.3 Modellierung logischer Prozesse

Bei der Simulation großer Modelle, die aus einer sehr großen Anzahl recht kleiner Objekte bestehen, auf einer beschränkten Anzahl von Rechnerknoten stellt sich die Frage, wie einzelne logische Prozesse in einer praktischen Implementierung realisiert werden.

6.3.3.1 Multithreading und Scheduling

Der direkte Ansatz ist die 1-zu-1-Abbildung der Objekte auf einen klassischen logischen Simulationsprozeß (LP) mit eigener Ereignisliste, Zustand, Simulationsuhr und sonstigen Verwaltungsstrukturen. Ein LP kann dabei als Thread des Betriebssystems oder einer simulationsspezifischen Zwischenschicht modelliert werden. Das Einplanen und die Auswahl des aktiven Threads erfolgt durch einen Scheduler, der verschieden implementiert sein kann. Lin zeigt verschiedene Ansätze dafür auf [LIN92a]. Nachteil dieses Verfahrens ist der Schedulingaufwand für die Organisation der Threads, der sich aber durch die leichtgewichtige Struktur (im Gegensatz zu Prozessen) noch kontrollieren läßt. Es stellt sich jedoch ebenfalls die Frage nach der Skalierbarkeit, die z.B. unter UNIX bei schwergewichtigen Prozessen ja schon mit einigen 10 LPs pro Rechner spätestens an ihre Grenzen stößt. Für Simulationen, die ihre LPs als Threads verwalten, wurden bisher auch nur kleinere Modelle untersucht. Betrachtungen von Warteschlangennetzsimulationen ergaben selbst auf einer speziell für schnelles hardwaremäßig unterstütztes Prozeßscheduling ausgelegten Transputerarchitektur Probleme mit der Leistungsfähigkeit bei größeren Modellen [MEI91a, RIC91a, MMR93a]. Außerdem fällt ein weitaus höherer Speicherbedarf zur Realisierung der einzelnen LPs an.

6.3.3.2 Verschmelzung von LPs

Der von uns gewählte Ansatz besteht in der vollständigen Verschmelzung der Datenstrukturen der LPs einer Partition. Dadurch läuft in der Regel nur ein einziger Simulationsprozeß auf einem Rechner, der alle LPs einer Partition beinhaltet. Indem alle LPs dieselbe Ereignisliste benutzen, werden die Events implizit in ihrer zeitlichen Reihenfolge sortiert und ein Scheduling einzelner LPs anhand minimaler Simulationszeit entfällt (Abb. 6.5). Weiterhin spart man sich die Schedulingaufrufe sowie die Synchronisation der Zugriffe auf gemeinsame lokale Datenstrukturen (z.B. Eingangs- und Ausgangskanäle), da es keine konkurrenten Zugriffe mehr gibt.

Eingangs- und Ausgangskanäle zwischen lokalen LPs auf demselben Prozessor existieren nicht mehr explizit. Die Kanalzeiten sind in die zeitliche Reihenfolge der Ereignisliste eingebettet. Da die LPs einer Partition verschmolzen sind, läßt sich dieses Vorgehen auch auf die Kanäle zwischen den Simulatoren ausweiten (Abb. 6.4). Zwischen zwei Partitionen wird somit nur noch je ein Kanal für (Ereignis-)Nachrichten pro Kommunikationsrichtung benötigt. Die Ereignisse aller betroffenen LPs laufen darüber und insbesondere bei (für konservative Verfahren unabdingbarer) FIFO-Kommunikation in aufsteigender zeitlicher Reihenfolge.

Allerdings weist diese Clusterung auf den ersten Blick auch einige Nachteile auf. Bei konservativen Verfahren könnte das Potential der Parallelbearbeitung von LPs eingeschränkt werden. Exi-

stiert eine Kette elementarer LPs, die von einer Quelle getrieben werden, die selbst nur sehr langsam Ereignisse produziert, so müssen viele nachfolgende Cluster blockierend warten, bis die Ereignisse dieser wenigen LPs eintreffen, obwohl die elementaren LPs genügend ausführbare Ereignisse hätten. Dieses Problem läßt sich jedoch durch geeignete Partitionierung ggf. mildern. Außerdem weist Fujimoto darauf hin, daß sich parallele Simulation erst dann rentiert, wenn alle LPs stets genügend Arbeit haben und ein entsprechend hoher Nachrichtenfluß vorhanden ist (Message avalanche), um die Verfügbarkeit einer ausreichenden Anzahl von Ereignissen zu garantieren [FUJ88b].

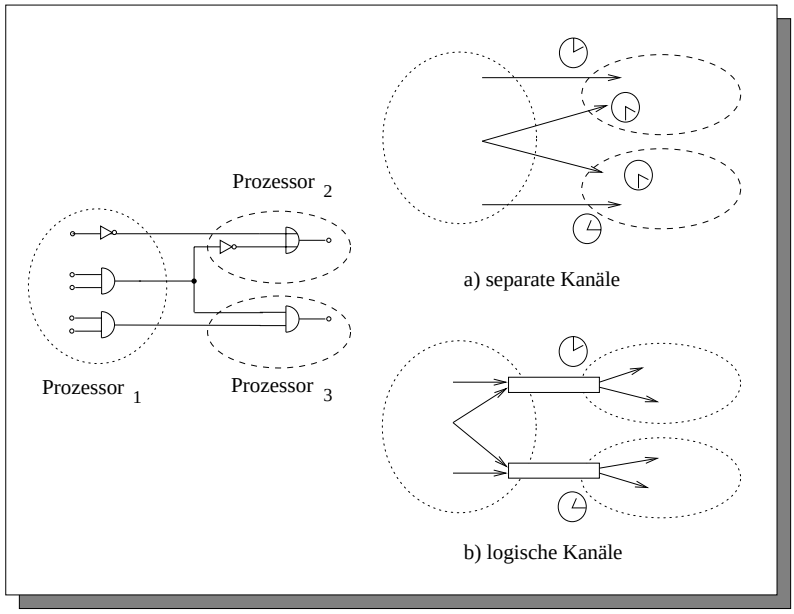


Abbildung 6.4: Organisation logischer Kanäle

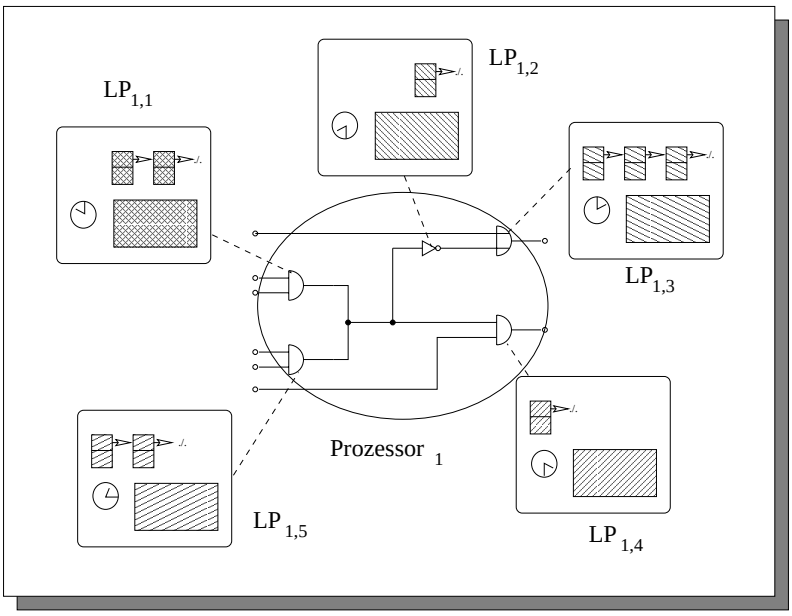
Das gilt insbesondere für das genannte Beispiel. Ein gutes Partitionierungsverfahren integriert die LPs einer solchen Kette in eine Partition. Selbst bei Verwendung elementarer LPs wird eine niedrige Aktivitätsrate im Modell zu einer verstärkten Blockierung eines großen Teils der LPs führen.

Existieren stets genügend ausführbare Ereignisse, so sollten sich beide Vorgehensweisen ähnlich verhalten. Die geclusterte Version weist dabei aber einen wesentlich geringeren Verwaltungsoverhead auf. Außerdem kann in einer threadbasierten Umgebung stets auch nur ein Thread zu einem Zeitpunkt aktiv sein. Bei Verwendung von multitasking-fähigen Rechnern oder Mehrprozessorarchitekturen besteht die Möglichkeit, auch mehrere Partitionen auf einen Rechnerknoten abzubilden, um dadurch ein gewisses Maß an potentiell erhöhter Parallelität im Sinne elementarer LPs zu bieten. Allerdings wechselt man dabei in der Regel aus dem Bereich der billigen Kontextwechsel bei Threads in eine Umgebung mit komplexen (und im UNIX-Bereich doch recht aufwendigen) Prozeßwechseln, so daß diese Vorgehensweise nur beschränkt eingesetzt werden sollte.

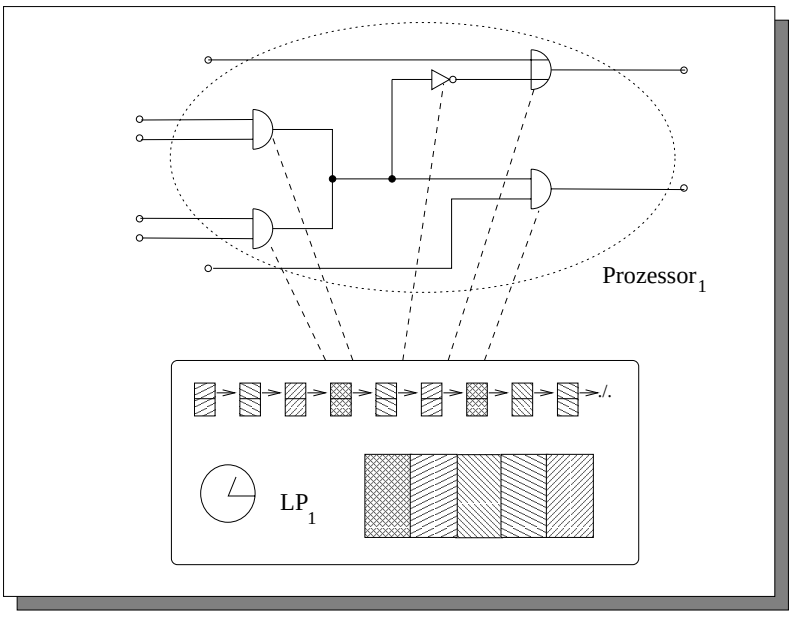
In einer optimistischen Synchronisationsumgebung treten die Probleme mit überflüssigen Blockaden nicht auf. Es stellt sich jedoch eine andere Frage. Wie soll die Sicherung des Zustands der elementaren LPs und insbesondere die Restauration im Falle eines Rollbacks erfolgen? Briner unterscheidet dabei nach *Component-synchronization* und *Processor-synchronization* [BRI90a].

Die Art und Weise der Zustandssicherung ist dabei unkritisch. Elementarer ist die Analyse, auf welche LPs sich ein Rollback bezieht. Bei der komponentenweisen Variante werden nur die Ereignisse des betroffenen LPs zurückgesetzt, korrigiert und zugehörige Antinachrichten verschickt.

Viele Partitionierungsmethoden gruppieren jedoch stark voneinander abhängende LPs innerhalb einer Partition. Deshalb wird eine Component-synchronization mitunter auch viele sekundäre, lokale Rollbacks bei anderen LPs auslösen. Das Verhalten ähnelt dadurch ohnehin der Processor-synchronization.



a) Ein LP pro Objekt



b) Ein LP pro Prozessor

Abbildung 6.5: Modellierung von LPs

Für DVSIM wurde die Prozessorsynchronisation gewählt. Die Verschmelzung einzelner LPs für DVSIM wurde zusammen mit den Arbeiten am parallelen Warteschlangensimulator DISQUE ent-

wickelt [RIC95a]. Tritt ein Rollback auf, so wird der gesamte Teilsimulator zurückgesetzt, wodurch eventuell auch einige nicht benötige Rücksetzungen ausgeführt werden. Man erspart sich jedoch die Analyse, für welche LPs und welche Ereignisse die Rücksetzungen zu erfolgen haben. Externe Auswirkungen "fehlerhafter" Zurücksetzungen können durch die Lazy-cancellation-Optimierung vermieden werden.

6.3.4 Andere Clusteransätze

Su und Seitz [SUS89a] schlagen eine komplette Zusammenfassung der LPs eines Prozessors unter Verwendung einer einzigen Ereignisliste zur Optimierung konservativer Simulationsverfahren (Reduktion der Nullnachrichten zwischen prozessorlokalen LPs) vor. Diese Partitionierung gehorcht dabei einem azyklischen Prinzip. Die Untersuchungen erfolgten auf einem simulierten Simulator unter der Annahme von einheitlichen feingranularen Ereignisbearbeitungskosten und unter Vernachlässigung der Kommunikation. Su und Seitz beschreiben dabei bei Prozessoranzahlen bis 2000 PEs nahezu lineare Beschleunigung. Bei bis zu 16 Knoten gibt es jedoch (nach Ansicht der Autoren) einige initiale Startprobleme, so daß dort zunächst keine Beschleunigungen gegenüber einem sequentiellen Simulator beobachtet werden. Daneben wurden auch Messungen auf iPSC/1 und iPSC/2-Rechnern gemacht und für einen kleinen Schaltkreis mit 1376 Gattern eine Beschleunigung von maximal 8 auf 128 Prozessoren beobachtet. In den von Briner für Time Warp untersuchten Modellen erwies sich keines der beiden Koppelungsextrema (processor oder component) für alle Modelle überlegen [BRI90a].

Avril und Tropper fanden bei der Kombination von komponentenbezogenem Checkpointing und Rollback eine bessere Leistungsfähigkeit als bei dem prozessororientierten Ansatz für zwei große ISCAS-Schaltungen. Allerdings verwenden sie keine vollständige Verschmelzung der LPs [AVT95a]. Die separaten Ereignislisten und Kanäle für die lokalen LPs jedes Prozessors bleiben erhalten. Lediglich die globale Koppelung mit LPs auf anderen Prozessoren erfolgt über gemeinsam verwaltete Kanäle und folgt dem Time Warp-Ansatz.

Schlagenhaft, Ruhwandl und Sporrer gehen einen zweistufigen Mittelweg, indem ca. 100 bis 1000 LPs zu einem Cluster mit gemeinsamer Ereignisliste zusammengefaßt werden. Mehrere solcher Cluster bilden dann eine Partition [SRS95a]. Alternativ kann zunächst auch eine Partitionierung erfolgen und anschließend erst die Cluster innerhalb jeder einzelnen Partition gebildet werden. Durch die Cluster sollen die Auswirkungen der Rollbacks auf die Gesamtpartition begrenzt werden.

6.4 Ablaufumgebung

DVSIM wurde für Rechnerarchitekturen mit physikalisch verteiltem Speichermodell entwickelt. Als Abstraktionsebene zur Programmierung der logischen Prozesse wird auf der Kommunikationsbibliothek PVM (Parallel-virtual-machine)[5] [SUN90a] aufgesetzt, die auf einer Vielzahl von Plattformen und unter anderem auch auf Shared-memory-Architekturen verfügbar ist. Dadurch erhöht sich die Portabilität des parallelen Simulator.

Die ersten verwendeten Plattformen waren ein Intel iPSC/860-Hypercube mit 32 Knoten sowie ein Netzwerk von Sun-Workstations. Mit wenig Aufwand wurde DVSIM auf einen massiv parallelen Rechner auf Transputerbasis (Parsytec GCel) mit 1024 Prozessoren sowie einen auf dem PowerPC aufsetzenden PowerXplorer (64 Knoten) am Paderborn Center for Parallel Computing portiert. Die Transputervariante konnte allerdings wegen der geringen Speicherausstattung von nur 4MB pro Prozessor nicht für größere Modelle genutzt werden.

[5] Die erste verfügbare DVSIM-Version lief unter PVM 2.4. Zur Zeit wird Version 3.3.11 unterstützt.

Außerdem existiert eine rudimentäre Anpassung für die prototypische SB-PRAM-Maschine [FGK97a]. Hier wurde jedoch das PVM stark ähnelnde p4-System[6] [BUL92a] verwendet, indem einfach die Bibliotheksaufrufe für die Kommunikation ersetzt wurden. Untersuchungen konnten jedoch aufgrund der instabilen Laufzeitumgebung auf dem Experimentalsystem nicht erfolgreich durchgeführt werden.

6.5 Implementierte Algorithmen in DVSIM

Zur Untersuchung des Parallelisierungspotentials wurden aus den typischen Kategorien der asynchronen Simulationsalgorithmen je ein Vertreter implementiert, dessen Verhalten jedoch mit einer Vielzahl von Parametern konfigurierbar ist. Weiterhin wurden die wichtigsten in der Literatur beschriebenen Partitionierungsverfahren realisiert, um eine Einschätzung ihrer Leistungsfähigkeit in Zusammenarbeit mit den Synchronisationsverfahren vornehmen zu können. Die realisierten Methoden werden in den folgenden Abschnitten zusammen mit den verwendeten Parametern beschrieben.

6.5.1 Synchronisationsverfahren

6.5.1.1 Konservative Methoden

Der implementierte Vertreter zu Deadlock-detection und -recovery beruht auf dem Grundverfahren von Misra [MIS86a]. Die Simulatoren laufen in einen globalen Deadlock und bei völligem Stillstand des gesamten Systems wird die Verklemmung mit einem Zwei-Wellen-Algorithmus erkannt [MAT89d]. Der Algorithmus läuft transparent und zeitparallel mit der eigentlichen Simulation. Wechselt ein Simulator in den passiven Zustand, weil er keine weiteren Ereignisse zu verarbeiten hat oder weil die Simulation aufgrund mangelnder Garantien nicht fortgesetzt werden kann, so wird das Token zur Erkennung eines Deadlocks unter Weitergabe der Anzahl lokal gesendeter und empfangener Nachrichten sowie des Zeitstempels des kleinsten lokal vorhandenen Ereignisses an einen festgelegten Nachbarn eines logischen Rings weitergereicht.

Ein Simulator im Ring übernimmt die Funktion des Controllers. Weisen die Summen der Sende- und Empfangszähler für zwei aufeinanderfolgende Besuche beim Controller dieselben Werte auf, so befindet sich das System in einem globalen Deadlock. Das Minimum aller lokalen Zeitstempel ist in diesem Fall eine untere Schranke für sicher ausführbare Ereignisse. Bei der nächsten Runde des Tokens werden diejenigen Simulatoren gestartet, deren kleinstes Ereignis mit dem Minimum übereinstimmt. Der Deadlock ist somit gebrochen und die Simulation kann fortgesetzt werden.

Als Optimierung kreist das Token nicht ständig, sondern verbleibt auf einem aktiven Simulator, bis dieser in den passiven Zustand wechselt.

6.5.1.2 Optimistische Methoden

Die Time Warp-Implementierung basiert auf dem Grundverfahren von Jefferson [JEF85a]. Erweitert wurde es um die Lazy-cancellation-Variante, die mit der Kompileroption LAZY_CANCELLATION eingeschaltet wird. Defaultmäßig verwendet DVSIM Lazy-cancellation.

Der Optimismus, mit dem ein Simulator in seiner Arbeit fortschreiten kann, wird durch ein statisches bzw. für jeden Simulator dynamisch zur Laufzeit ermittelbares Fenster gesteuert. Die initiale Fenstergröße wird der Variablen DVSIM_TW_BTW_SIZE entnommen. Die Berechnung der neuen Fenstergröße erfolgt alle n Rollbacks. Treten viele Rollbacks auf, so erfolgt eine häufige

[6] PVM entstand ursprünglich aus dem p4-System und einigen anderen Kommunikationsbibliotheken

Anpassung. Ist die Simulation im Sinne von Rollbackminimierung stabil, so wird die Aktualisierung seltener sein. Der Wert von n kann über die Umgebungsvariable DVSIM_TW_BTW_UPDATE kontrolliert werden. Treten zwischen zwei Aktualisierungen der GVT keine Rollbacks auf, so war die Fenstergröße offenbar gut gewählt und es erfolgt eine Verdoppelung. Sind dagegen Rollbacks erfolgt, stehen zwei Verfahren zur Fensterverkleinerung bereit.

- Die neue Fenstergröße ergibt sich aus Simulationsfortschritt zum Zeitpunkt eines Rollbacks rb $advance_{rb} = LVT - GVT$ (Differenz zwischen LVT und GVT) und der Tiefe des Rollbacks $depth_{rb} = LVT - t_{straggler}$. Die Differenz beider Werte ergibt die neue Breite. Um lokale Ausreißer auszugleichen, erfolgt die Berechnung über Mittelwerte der letzten n Rollbacks. Die Differenz entspricht in etwa dem Simulationszeitraum, in dem sinnvolle Arbeit geleistet wurde, so daß der Simulator danach prinzipiell auch blockiert werden kann. Nach Festlegung eines neuen Wertes startet die Aufzeichnung wieder neu ohne Berücksichtigung der bisherigen Historie. Es ergibt sich somit folgende Berechnungsvorschrift der Fenstergröße:

$$cur\_btw\_size = \frac{1}{n}\sum_{i=1}^{n} advance_{rb_i} - \frac{1}{n}\sum_{i=1}^{n} depth_{rb_i}$$

  Das Fensters darf eine in der Variablen DVSIM_TW_MIN_BTW_SIZE festgelegte Mindestgröße nicht unterschreiten, und die Änderung muß eine gewisse Relevanz haben. Die Schwelle hierfür wird in DVSIM_TW_MIN_RB_DEPTH eingestellt.

- Die zweite Anpassungsweise berücksichtigt das Verhältnis der effektiv zwischen Rollbacks ausgeführten und nicht zurückgesetzten Ereignisse zur Anzahl aller Ereignisse. Die Fenstergröße wird mit diesem Faktor multipliziert und dadurch ggf. verkleinert:

$$cur\_btw\_size = cur\_btw\_size \times \frac{1}{n}\sum_{i=1}^{n} \frac{effective\_events_i}{all\_events_i}$$

Die Kompilerflags BTW_TIME_DIFF bzw. BTW_EVENT_DIFF legen fest, nach welcher Metrik die Fenstergröße bei dynamischer Anpassung neu bestimmt wird. Durch geeignete Wahl der Parameter erhält man somit unterschiedliche Umsetzungen des Time Warp. Setzt man die minimale Fenstergröße DVSIM_TW_MIN_BTW_SIZE auf einen sehr großen Wert (virtuell unendlich), so wird der Grundalgorithmus ausgeführt. Hat die Aktualisierungsrate DVSIM_TW_BTW_UPDATE einen sehr großen Wert, so wird mit der statischen Fensterbreite DVSIM_TW_BTW_SIZE gearbeitet, die durchaus auch auf den Wert Null gesetzt werden kann. In diesem Fall resultiert daraus ein Verfahren, das dem Local-time-warp-Approach von Rajaei, Ayani und Thorelli ähnelt [RAT93a].

Die Überlegungen zur Regelung der Fenstergröße sind mit Ansätzen vergleichbar, die in Datennetzen zur Überlastregelung (Congestion-control) eingesetzt werden. Ein günstiges Verhalten wird belohnt, indem der zulässige Optimismus erhöht wird. "Fehlfunktionen" werden durch Fensterreduktion bestraft (vgl. Slowstart-Verfahren bei TCP [PED96a]).

Die Häufigkeit der GVT-Berechnung wird durch die Variable DVSIM_TW_GVT_DELAY reguliert und von einem beliebigen aber festen Simulator kontrolliert. Die Initiierung einer neuen GVT-Ermittlung geschieht in Abhängigkeit von der Anzahl der Simulationsschritte beim Initiator. Diese Kontrolle erfolgt bei der Anfrage des Simulators zur Freigabe der Simulation im SIM_FILTER. Wurden seit Ende der letzten GVT-Berechnung DVSIM_TW_GVT_DELAY Simulationszyklen (Ereignisse) ausgeführt, startet eine neue Runde. Die Defaulteinstellung ist 50 Aufrufe. Außerdem existieren noch drei weitere Optimierungsflags für:

- die Speicherverwaltung häufig benötigter Datenstrukturen über selbst organisierte Freispeicherlisten, so daß Allokation und Freigabe nicht die Laufzeit dominieren (TWMSG_RECYCLE),
- den Zugriff auf die projektierten Signalfolgen der einzelnen Knoten. Die Liste der Ereignisse wird in je eine eigene Struktur für bearbeitete und unbearbeitete Ereignisse zur schnelleren Realisierung von Such-, Einfüge- und Löschoperationen aufgeteilt (PROCESSED_EVS),
- einen optimierten Suchalgorithmus auf der Liste der Eingangsnachrichten, die als Skip-list realisiert wurde [PUG90a] (SLIST_REDUCED_COMPARISONS).

Der Versuch, die Kontrolle einer globalen Speicherverwaltung über das Cancelback-Protokoll zu realisieren, scheiterte an dem unterliegenden GVT-Berechnungsalgorithmus, der die in einem System mit nichtatomarem Nachrichtenversand zurückgeschickte Nachrichten bei der GVT-Berechnung nicht korrekt registrieren kann und deshalb falsche Werte bei Aktivierung des Cancelback-Moduls liefert. Die Verfügbarkeit einer effizienten Realisierung von GVT-Algorithmen bei Verwendung des Cancelback-Verfahrens ohne Einfrieren erscheint fraglich.

6.5.1.3 Protokolle zur Leistungsbewertung anderer Verfahren

Das Oracle-log-Verfahren [SWF87a] wurde in das konservative Deadlock-detection-Protokoll integriert. Nach Angabe der zu verwendenden Log-Dateien als Parameter des Befehls `oracle` (an der Bedienschnittstelle von DVSIM) wird automatisch auf die gespeicherten Daten zugegriffen anstatt das konventionelle Protokoll zu verwenden.

Das Einlesen der Log-Daten benötigt Rechenzeit und Speicherplatz. Um den Startup zu beschleunigen und die Belastung des Speichers zu reduzieren, erfolgt das Einlesen blockweise. Die Anzahl gleichzeitig geladener Logs wird durch die Konstante MAXORACLE (default 1024) bestimmt.

Der Empfang des zu einem Protokolldatensatz gehörigen Ereignisse bewirkt, daß der Eintrag aus der Liste der erwarteten Ereignisse gelöscht wird. Sind keine Einträge mehr in der Liste vorhanden, wird ein neuer Block nachgeladen, bis schließlich das Dateiende erreicht wird. Die Simulationsfreigabe im SIM_FILTER erfolgt dann, wenn das Orakel nur noch Einträge mit größeren Zeitstempeln als der angefragte Zeitpunkt enthält.

Die verteilte Variante kann sowohl das Binär- als auch das Textformat (PLAIN und VERBOSE) verarbeiten. Da es sich um einfache Leseoperationen handelt, ist die Bearbeitungsgeschwindigkeit beider Datenarten in etwa gleich. Die Formate dienen lediglich Effizienzgesichtspunkten bezüglich des Speicherbedarfs auf dem Dateisystem.

6.5.1.4 Steuerungsparameter für alle Verfahren

Als allgemeiner Parameter für alle Verfahren kommt zum einen die Kontrollvariable für die Ereignisgranularität zum Einsatz. Der Kompilerschalter ARTIFICIAL_DELAY aktiviert das Modul zur Steuerung der Granularität einzelner Ereignisse. Er wurde eingeführt, um den Overhead, der auch bei Erhöhung der Granularität um null Zeiteinheiten durch die algorithmische Erweiterung anfällt, für die Standarduntersuchungen mit elementaren Ereignissen ohne künstliche Verlängerung komplett abschalten zu können. Die Umgebungsvariable DVSIM_EVAL_DELAY legt die exakten Werte der Granularitätserhöhung in Mikrosekunden fest.

Das Auslesen neu angekommener Nachrichten besitzt stets die höchste Priorität gegenüber anderen Aktionen des Simulators. Nach jedem Simulationszyklus, der aus der Auswertung aller vorgemerkten Objekte, dem Fortschalten der Simulationsuhr und der Einplanung neuer Ereignisse

zu einem Simulationszeitpunkt besteht, wird nachgeschaut, ob neue Nachrichten vorliegen, und vorhandene Nachrichten werden zunächst in eine Input-Queue eingekettet. Durch eine Konstante MSG_MAX wird die Anzahl der in der Queue zwischengespeicherten Nachrichten defaultmäßig auf 512 begrenzt. Übersteigt die Anzahl der wartenden Nachrichten diese Schranke, so bleibt die Pufferung der restlichen Nachrichten dem Kommunikationssystem überlassen. Die Kompileroption PRIO_QUEUE regelt zusätzlich, ob für Nachrichten mit unterschiedlichen Prioritäten jeweils eine eigene Warteschlange zum schnelleren Zugriff verwendet wird.

Eine weitere Option XDR legt fest, ob die von der Kommunikationsbibliothek PVM verschickten Nachrichten in die rechnerunabhängige External-data-representation (XDR) konvertiert werden müssen, weil Rechner mit heterogenen internen Datenformaten bei der Simulation zusammenarbeiten.

Schließlich lassen sich bei aktiviertem Kompilerschalter TRACE Tracedaten zu Simulationsablauf sowie Nachrichtenaufkommen und -inhalten protokollieren, die entweder mit dem allgemeinen Werkzeug ParaGraph [HEF91a] oder den simulationsspezifischen Visualisierungstools YES [STU93a] bzw. INVADES [MAR95a] aufbereitet werden können. Zur Garantie der korrekten kausalen Beobachtbarkeit der Ereignis- und Nachrichtenreihenfolge benötigt man jedoch einen Satz genau gehender Hardwareuhren auf den Prozessorknoten. Die Uhren dürfen höchstens eine lineare Drift aufweisen, um durch eine Nachsynchronisation der Traces die kausalen Abhängigkeiten durch eine totale Ordnung auf den Zeitstempeln repräsentieren zu können. Dazu ist eine Synchronisation der verteilten Uhren zu Simulationsbeginn und -ende zur Ermittlung der Drift nötig [MEI91b]. Die Forderung nach linearer Drift wird von den verwendeten Sun-Workstations leider nicht erfüllt, so daß mitunter selbst nach der Korrekturphase auch Nachrichten als "in die Vergangenheit verschickt" erscheinen.

6.5.2 Partitionierungsverfahren

Die verschiedenen Partitionierungsansätze wurden bereits grundsätzlich in Kapitel 5 behandelt. Die in DVSIM realisierten Verfahren werden hier nur kurz genannt und ggf. besondere Parameter beziehungsweise Ausprägungen der Verfahren erwähnt.

- Round-robin dient als einfache Variante als Vertreter der Klasse der zufallsbasierten Verteilungen.
- Die Stringpartitionierung erfolgt wie bereits beschrieben. Die Länge der Strings wird dabei kontrolliert und darf den Wert $\lceil \frac{n}{k} \rceil$ bei n Objekten und k Partitionen nicht überschreiten, damit in etwa balancierte Partitionen bezüglich der zugewiesenen Objektanzahl gebildet werden können. Die Zuweisung an verschiedene Partitionen erfolgt nach der rangweisen Sortierung der Startknoten der Strings (wie bei der Natural-Verteilung beschrieben). Dieselbe Zuordnungsmethode findet sich bei Cones- und zyklenfreier Partitionierung.
- Cones werden als Fan-in- und Fan-out-Kegel verwendet. Allerdings wird auch hier eine Balancierung der Partitionsgröße anhand der Objektanzahl angestrebt. Weiterhin werden einzelne Objekte nur genau einem Kegel zugeordnet und somit Überlappungen vermieden. Die Ermittlung der Cones erfolgt in einer Greedy-Tiefensuche, so daß im Extremfall ein Kegel stark in Richtung eines Strings entarten kann. Spezielle Startpunkte neben den Aus- oder Eingängen der Schaltung (z.B. Flipflops) werden aufgrund der flachen Modellstruktur nicht gewählt. Daneben wurden aber auch Partitionierungsverfahren mit unlimitierter Breitensuche für Fan-in- und Fan-out-Kegel realisiert. Die Größen einzelner Kegel können hierbei stark voneinander abweichen.

- Die zyklenfreie Partitionierung ähnelt sehr stark der Cone-Partitionierung. Jedoch wird die Größenbeschränkung der starken Zusammenhangskomponenten nicht eingeführt, damit alle Objekte eines Zyklus derselben Partition zugewiesen werden. Dadurch können stark ungleichgewichtige Aufteilungen entstehen.
- Für Kernighan/Lin findet zunächst das sequentielle Abtrennen von k Partitionen statt. Daran schließt sich die paarweise Optimierung der gefundenen Partition an. Die Kantengewichte sind identisch. Durch ein Ansimulieren der Schaltung können aber realistischere Werte bezüglich der Anzahl der verschickten Ereignisnachrichten ermittelt werden. Die Schaltung kann anschließend eingesammelt, erneut partitioniert und mit der hoffentlich optimierten Aufteilung erneut gestartet werden.
- Die Kantengewichte bei Soccer werden analog zum Verfahren bei K/L gesetzt.

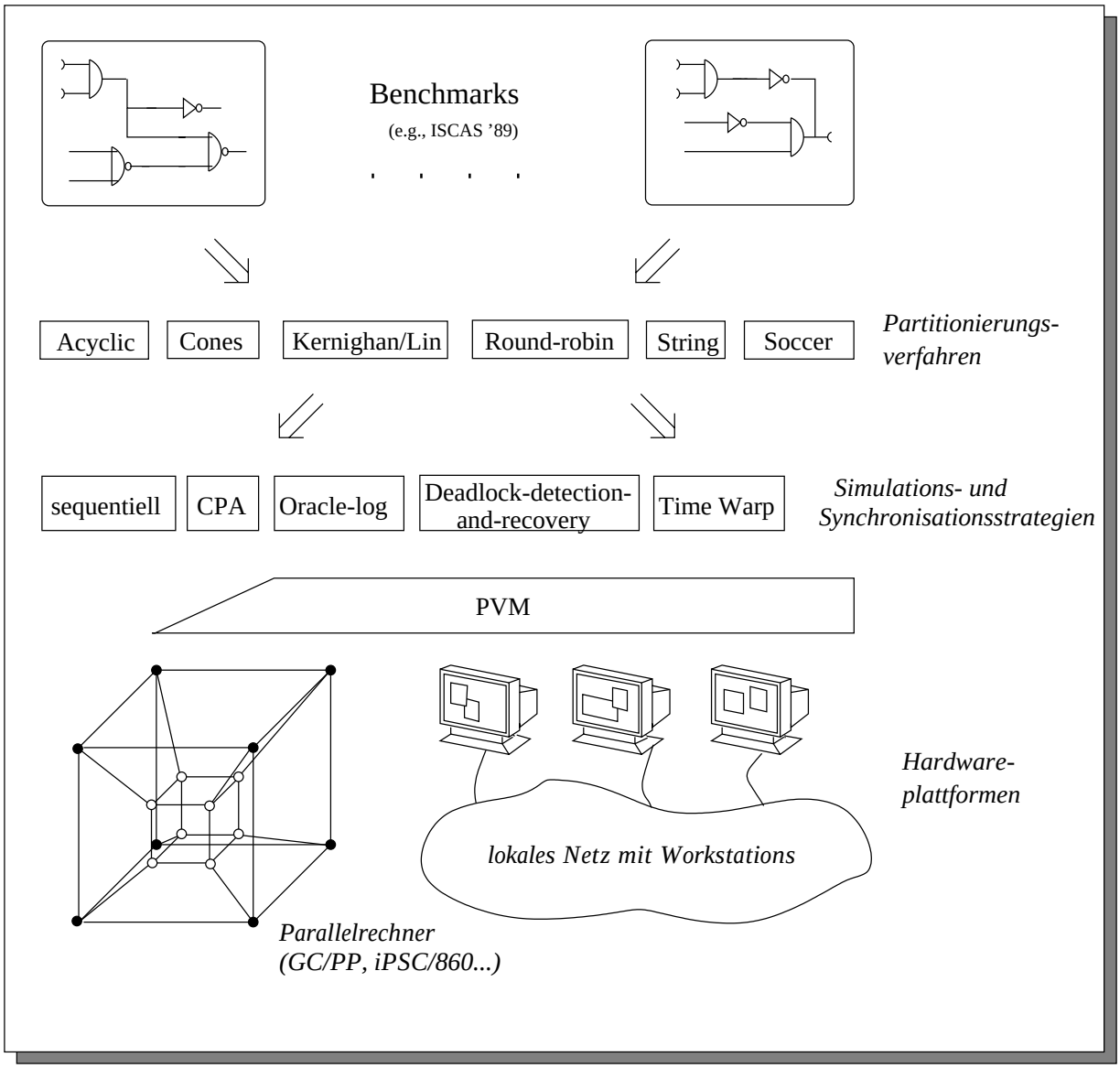


Abbildung 6.6: Komponenten der Evaluierungsumgebung

Insgesamt ergibt sich somit das in Abb. 6.6 gezeigte Szenario mit Partitionierungsalgorithmen, Mapping- und Verteilungsfunktionalität, Synchronisationsverfahren und Hardware-Ablaufumgebung.

In den folgenden Kapiteln werden nun die eigentlichen Untersuchungen, die mit Hilfe dieser Umgebung durchgeführt wurden, beschrieben, die Ergebnisse diskutiert und versucht, eine Übertragung auf das allgemeine Feld der PDES vorzunehmen.

Kapitel 7

Bewertungsgrundlagen für Simulationsverfahren

Dieses Kapitel legt die Basis für die Evaluierung paralleler Simulationsansätze. Zunächst wird die für die sequentiellen und parallelen Untersuchungen verwendete Hard- und Softwareumgebung mit ihren Leistungsparametern beschrieben. Anschließend erfolgt die Präsentation verwendeter realer und künstlicher Benchmarks. Die erste Gruppe besteht aus Schaltungen der ISCAS'89-Sammlung [BBK89a], während die zweite typische für die parallele Simulation kritische Topologien wie Modelle mit Zyklen oder einfache Pipelines nutzt, um die Grenzen der Leistungsfähigkeit der Verfahren auszuloten. Die sequentiellen Laufzeiten der einzelnen Modelle werden als Vergleichsbasis für die parallelen Untersuchungen präsentiert. Die Beleuchtung der Beschleunigungpotentiale mittels CPA beendet diesen Abschnitt und berücksichtigt dabei die wichtigen Faktoren Ereignisgranularität und Nachrichtenlaufzeit.

7.1 Simulationsumgebung

7.1.1 Meßgrößen

Die Evaluierung und Bewertung der Leistungsfähigkeit paralleler diskreter ereignisgesteuerter Simulationsverfahren erfolgt im folgenden einerseits anhand gemessener Laufzeiten und andererseits über hieraus abgeleitete Größen wie die Länge des kritischen Pfads der Berechnung. Die Basis dafür sind die real verstrichenen Zeiten, die ein Anwender bei der Beurteilung ebenfalls zugrundelegen würde. Die Messungen erfolgen mit Systemaufrufen, die auf die Realzeituhren der Rechner zugreifen. Gelegentlich wird, wo es sinnvoll erscheint, auch die Anzahl der bearbeiteten Ereignisse als Vergleichsmaßstab herangezogen.

7.1.2 Rechnerplattform

Die beschriebenen Untersuchungen beziehen sich zunächst auf Experimente, die auf einem Parsytec GC/PP-Parallelrechner[1] am Zentrum für Paralleles Rechnen (PC^2) der Universität/GH Paderborn durchgeführt wurden. Aspekte der Übertragbarkeit auf ein Workstation-Cluster werden in Kapitel 10 behandelt.

Der verwendete Parallelrechner verfügt über 64 Prozessoren vom Typ Motorola PowerPC 601, die mit 80 MHz getaktet werden und über 64 MB Hauptspeicher für jeweils 2 auf einem Board

[1] GrandChallange PowerPlus

untergebrachte Prozessoren verfügen. Das Kommunikationsnetz wird von T800-Transputern realisiert, die in Form eines Gitters mit 4 Transputerlinks pro "Himmelsrichtung" als FatMesh verschaltet sind. Es können bis zu 20 Mbit Daten pro Sekunde in jeder Richtung übertragen werden. Dazu verfügt jedes Board über 4 Transputer, die sich den gemeinsamen Hauptspeicher mit den Prozessoren teilen und Daten per DMA mit den Prozessoren austauschen [SIM94b]. Die PowerPC-Prozessoren werden dadurch im wesentlichen von Belastungen durch Kommunikationsaufwand und Routingaufgaben befreit.

Als Betriebssystem ist das proprietäre aber POSIX-konforme UNIX-Derivat PARIX[2] [PAR95a] des Herstellers Parsytec installiert. PVM ist in der Version 3.2.6 vorhanden [PAR94a]. Die Latenzzeiten des Kommunikationsnetzwerks betragen im Mittel 276 μs für Initialisierung und Setup des Links sowie ca. 300 μs für den kompletten Versand einer 50 - 100 Byte langen Nachricht (typische Größe der verwendeten Ereignisnachricht) [KEL95a]. Die Granularität der primitiven Simulationsereignisse (Ereignislistenoperationen und Auswertung eines Gatters) liegt bei 10 – 12 μs bei Gattern mit zwei oder mehr Eingängen. Inverter benötigen zu Auswertung etwa die Hälfte dieser Zeit. Wir erhalten somit als Basis für das Verhältnis zwischen Versand einer Nachricht und Berechnungsdauer eines Ereignisses in einem realen Schaltkreis etwa den Faktor 30.

Die Hardwareuhr kann über Systemaufrufe mit zwei Auflösungen ausgelesen werden. Diese betragen entweder 1 oder 64 μs. Für vorliegende Untersuchungen wurde die feinere Variante verwendet[3]. Es wurde durch Einführung von Korrekturfaktoren weitgehend versucht, systematische Meßfehler, die durch die durchgeführten Messungen selbst erzeugt werden, zu eliminieren[4].

7.2 Benchmarks und Eingabedaten

Die verwendeten realen Schaltungen stammen aus der ISCAS-Benchmark-Suite von 1989 [BBK89a]. Weitere künstliche Modelltypen wurden existierenden Benchmarks aus dem Bereich der Simulation von Warteschlangennetzen nachempfunden.

7.2.1 Beschreibung der realen Schaltungen

Es handelt sich bei den ISCAS'89-Modellen um sogenannte sequentielle Schaltungen, die aufgrund vorhandener D-Flipflops mit Daten- und Takteingang (FF) und Rückkoppelungen mit einem Zustand behaftet sind. Der Zustand ist durch alle bis zum aktuellen Zeitpunkt ausgeführten Schaltvorgänge definiert. Die Größenordnung der Schaltungen ist in Tabelle 7.1 dargestellt. Die Flipflops werden zur Simulation in ihre elementaren Gatter aufgebrochen. Daneben besteht die Option, ein Flipflop als einzelnen VHDL-Prozeß zu modellieren. Dadurch ergeben sich zweierlei Gesichtspunkte. Das Aufspalten der FFs erhöht die Gatteranzahl um 9 weitere Elemente pro FF (insgesamt 10 Gatter). Dagegen erhält man bei Realisierung der FFs durch lediglich einen Prozeß Objekte mit unterschiedlicher Granularität und es existieren entsprechend weniger Objekte als im ersten Fall.

Für die Untersuchungen wurden drei verschiedene Schaltungen aus der Sammlung von Modellen ausgewählt. Einziges Kriterium war die Verwendung unterschiedlicher Modellgrößen. Konkrete Angaben zur Funktionalität der Schaltungen liegen ebensowenig vor wie Eingabemuster zur Überprüfung der Schaltfunktionen. Die Modelle wurden ursprünglich zur Ermittlung der Qualität von

[2] Parallel Extensions for UNIX, Version 1.3.1

[3] Da der PPC-Prozessor für die Uhr ein 32-Bit-Register verwendet, erfolgt etwa alle 71 Minuten ein Überlauf, der einen Erkennungsmechanismus zur besonderen Behandlung bei länger laufenden Prozessen erfordert.

[4] Bei allen Zeiträumen wird deshalb in der Regel die Dauer eines Systemaufrufs zur Ermittlung der aktuellen Zeit vom Meßwert abgezogen.

Algorithmen zur automatischen Generierung von Testmustern bei der Fehlersimulation bereitgestellt. Sie fanden jedoch ebenso wie ihre kombinatorisch[5] aufgebauten Vorgänger (ISCAS'85) schnell auch Aufnahme in der parallelen Simulationsgemeinde, da ansonsten nur wenige Benchmarks frei verfügbar sind. Die im folgenden untersuchten Schaltungen tragen die Bezeichnungen s1196, s13207 und s35932.

Name	Eingänge	Ausgänge	ISCAS-				VHDL-	
			Signale	D-Flip-flops	Gatter	Gatter-äquiv.	Signale	Gatter
s27	4	1	13	3	10	40	54	46
s208.1	10	1	112	8	104	184	230	216
s298	3	6	133	14	119	259	300	293
s344	9	11	175	15	160	310	382	369
s349	9	11	176	15	161	311	385	372
s382	3	6	179	21	158	368	445	438
s386	7	7	165	6	159	219	237	226
s400	3	6	185	21	164	374	457	450
s420.1	18	1	234	16	218	378	464	442
s444	3	6	202	21	181	391	496	489
s510	19	7	217	6	211	271	417	394
s526	3	6	214	21	193	403	473	466
s641	35	24	398	19	379	569	636	597
s713	35	23	412	19	393	583	673	634
s820	18	19	294	5	289	339	500	478
s832	18	19	292	5	287	337	498	476
s838.1	34	1	478	32	446	766	932	894
s953	16	23	424	29	395	685	954	934
s1196	14	14	547	18	529	709	910	892
s1238	14	14	526	18	508	688	902	884
s1423	17	5	731	74	657	1397	1579	1558
s1488	8	19	659	6	653	713	744	732
s1494	8	19	653	6	647	707	738	726
s5378	35	49	2958	179	2779	4569	5422	5383
s9234	19	22	5825	228	5597	7877	8563	8540
s13207	31	121	8620	669	7951	14641	15744	15709
s15850	14	87	10369	597	9772	15742	16966	16948
s35932	35	320	17793	1728	16065	33345	40724	40685
s38417	28	106	23815	1636	22179	38539	43006	42974
s38584	12	278	20705	1452	19253	33773	37378	37362

Tabelle 7.1: Eigenschaften der ISCAS'89-Schaltungen

- s1196 ist ein kombinatorischer Schaltkreis, in dem D-Flipflops an zufälligen Stellen eingefügt wurden.
- s13207 basiert auf einem realen Chip-Entwurf (ebenso wie s9234, s15850, s38417 und s38584).

[5] d.h. der Zustand hängt allein von den aktuell anliegenden Eingangssignalen ab

- Über s35932 sind keine weiteren Informationen vorhanden. Das Modell gehört zu den drei größten Schaltungen.

Die Beschreibung der Schaltungen lag in einem einfachen Textformat vor und wurde zur Verwendung mit DVSIM nach VHDL konvertiert[6]. Die Anzahl der bei der Umwandlung erzeugten VHDL-Signale[7] und -Objekte[8] ist ebenfalls in Tabelle 7.1 enthalten.

7.2.2 Beschreibung der künstlichen Schaltungen

Die synthetischen Modelle orientieren sich an verschiedenen Topologien. Sie sollen auch die Untersuchung der Skalierbarkeit der Simulationsverfahren für größere Modelle gestatten, da die realen Schaltungen mit maximal 41.000 Gattern nur eine moderate Größe erreichen. Wir verwenden als Schaltungsobjekte überwiegend Inverter, die eine hohe Aktivitätsrate und homogenes Schaltverhalten aufweisen. Untersucht wurden:

1. Ketten aus n Invertern, die miteinander verschaltet sind,
2. geschlossene Zyklen mit $n-1$ Invertern und einem AND-Gatter zur Koppelung der rückführenden Kante,
3. Fan-out-/Fan-in-Modelle, die in $(k-2) < n$ parallele Äste mit Ketten aus Invertern verzweigen. Diese Modelle besitzen außerdem eine Kombinationsstufe zur Zusammenführung der Zweige,
4. sowie eine Erweiterung der vorherigen Modelle um rückführende Kanten, die einerseits innerhalb der einzelnen Zweige und andererseits vom Primärausgang zum Primäreingang (wie unter 2.) über das gesamte Modell Rückkoppelungen schaffen.

Die künstlichen Modelle werden entsprechend der obigen Reihenfolge mit den Namen `invn_chain`, `invn_cycle`, `invn_k_inout` und `invn_k_cs` bezeichnet, wobei n für die Gesamtanzahl der im Modell vorhandenen Elemente steht, $k-2$ die Zahl der parallelen Verzweigungen angibt und `inv` gewählt wurde, da die Modelle weitgehend aus Invertern bestehen.

Am ersten Modelltyp soll die Ausnutzung des Parallelismus einer gefüllten Pipeline durch die Synchronisationsalgorithmen unter dem Einfluß verschiedener Partitionierungen untersucht werden. Die zweite Variante betrachtet zusätzlich die Auswirkungen der Rückkopplung auf die Leistung der parallelen Simulatoren. Für konservative Methoden wird hier eine starke Leistungseinbuße aufgrund massiv auftretender Deadlocks bzw. bei anderen konservativen Implementierungen durch eine steigende Anzahl von Nullnachrichten erwartet.

Die dritte Klasse erweitert die prinzipielle Unabhängigkeit der einzelnen Prozessoren voneinander und repräsentiert somit nur lose durch kausale Abhängigkeiten und Kommunikationskanäle

[6] Dazu wurde ein Konverter (isc2vhdl) entwickelt, der die VHDL-Entities und -Architectures, die die ISCAS-Objekte nachbilden, einer Bibliothek entnimmt und daraus eine komplette VHDL-Beschreibungsdatei generiert. Durch Modifikation der Bibliothek können VHDL-Modelle auf unterschiedlichen Ebenen modelliert werden (z.B. für variierende Realisierungen von Flipflops).

[7] Die Anzahl der VHDL-Signale berechnet sich aus der Anzahl der ISCAS-Signale, vier vordefinierten Signalen (Clock, Clear, Low, High), jeweils neun internen Signalen pro FF, Eingangs- und Ausgangssignale, sowie jeweils einem Invertertreiber pro NAND und NOR, da diese Gattertypen durch AND und OR mit nachgeschaltetem NOT realisiert wurden.

[8] Die Menge der VHDL-Objekte ergibt sich aus der Anzahl der elementaren ISCAS-Objekte, je 10 Gattern pro Flipflop, einem zusätzlichen Gatter pro Primärausgang und den Korrekturgrößen für NANDs und NORs.

gekoppelte Modelle. Hierbei wird insbesondere eine manuelle Partitionierung entsprechend der Topologie (Zusammenfassen aller Inverter einzelner Zweige in einem Simulator) anderen Verfahren gegenübergestellt, wie sie beispielsweise die balancierte Kegelpartitionierung liefert. Zur Optimierung der Lastbalancierung wird n als Vielfaches der Partitionsanzahl k gewählt[9].

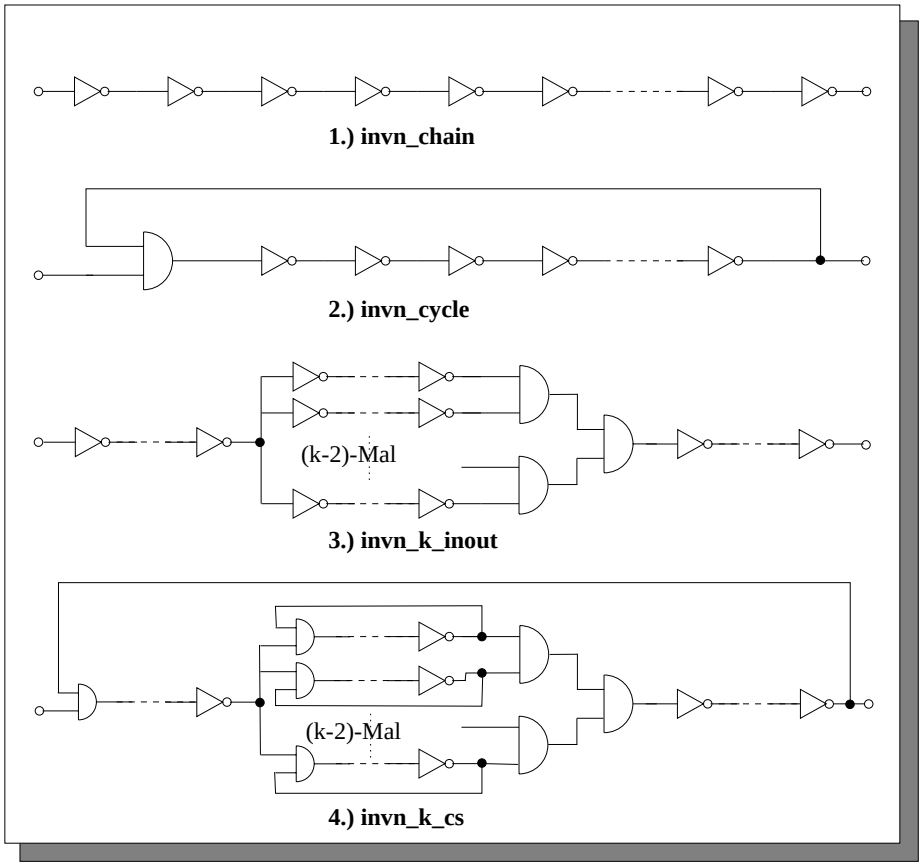


Abbildung 7.1: Schematische Darstellung der synthetischen Schaltungen

Die letzte Modellgruppe dient schließlich der Erhöhung der Komplexität bei Rückkoppelung der Elemente und somit der Steigerung des Kontrollnachrichtenaufkommens bei der parallelen Simulation. Sie ähnelt ein wenig dem Central-server-Ansatz bei Warteschlangennetzen.

7.2.3 Vergleichbarkeit mit Modellen anderer Forschergruppen

Viele Untersuchungen im Bereich der parallelen Simulation wurden anhand von Warteschlangenmodellen vorgenommen (siehe Kapitel 3). Diese können hier höchstens den künstlichen Modellen gegenübergestellt werden. Mit der Verwendung der ISCAS-Schaltungen folgen wir aber einem verbreiteten Ansatz, da sie von vielen anderen Forschern ebenfalls als Basis genutzt werden.

Allerdings werden oft die ISCAS'85-Schaltungen bevorzugt, da sie aufgrund weniger (oder keiner) Rückkopplungen ein besseres Parallelisierungsverhalten erreichen. Karthik und Abraham erzielen mit den kombinatorischen Schaltungen einen Speedup von 4 auf 5 Workstations während mit sequentiellen Benchmarks nur eine Beschleunigung von 2 auf 4 Prozessoren erreicht wird [KAA92a].

[9] Je eine Fan-out- und Fan-in-Partition und $k-2$ Zweige

Luksch beschränkt sich ebenfalls auf die Schaltungen der ISCAS'85-Gruppe. Er erreicht mit konservativen Verfahren für kleine kombinatorische Schaltungen Werte bis 2,5 auf 8 Prozessoren. Time Warp lieferte maximale Speedup-Werte von 5 auf 15 Knoten [LUK93a]. In der Regel lagen die Beschleunigungen für Time Warp zwischen 2 und 3 bei 15 Prozessoren.

Manjikian und Loucks [MAL93a] erzielten mit sequentiellen ISCAS-Schaltungen für konservativen Verfahren Werte von 3,7 auf 7 Rechnern und eine Beschleunigung von 4,1 auf 7 Knoten mit einem hybriden Time Warp, der Rollbacks auf die lokale Ausführung beschränkt. Dieselben Schaltungen verwendeten auch Müller-Thuns u.a. [MSD89a] sowie Bauer und Sporrer [BAS93a] mit ähnlichen Ergebnissen.

7.2.4 Aufbau und Länge der Stimulidaten

Da für die realen Schaltungen keine Angaben über ihre Funktionalität vorhanden sind, wurden für die Experimente zufällige Eingabeströme über dem Alphabet {0, 1} generiert, die an die Primäreingänge der Schaltung angelegt werden. Die Länge der Stimuli wurde so bemessen, daß die Modifikationen an den Eingängen sich durch Änderungen an allen Ausgängen bemerkbar machen. Diese Vorgehensweise wird prinzipiell von allen Nutzern der ISCAS-Schaltungen mehr oder weniger identisch angewandt. Durch Variation des Startwerts der Stimuliberechnung können unterschiedliche Eingabedaten generiert werden, um bereits für zufällig gewählte Eingaben überprüfen zu können, ob die Leistung der Simulationsverfahren von der konkreten Wahl der Stimuli abhängt. In diesem Falle wäre es extrem schwierig, überhaupt eine Vorhersage über die Laufzeit sequentieller Simulationen und insbesondere über die Beschleunigung bei paralleler Simulation zu treffen. Luksch fand, daß die Auswahl der Stimuliwerte von Relevanz bei kombinatorischen Schaltungen ist [LUK93a]. Manjikian und Loucks konnten keinen Einfluß bei der Simulation sequentieller Schaltungen entdecken [MAL93a].

Für jeden Eingang werden vom Stimuligenerator zunächst zufällige Folgen mit 32 Signalwechseln generiert. Werden längere Vektoren benötigt, so wird dieselbe zufällige Eingabefolge entsprechend oft wiederholt. Jeder einzelne Wert der Vektoren liegt für eine konstante Simulationszeit[10] am entsprechenden Eingang an.

Der Uhrtakt für die Flipflops wurde nicht betrieben. Er behält während der gesamten Simulationsdauer den Wert undefiniert ("u"), der von VSIM bei Auswertungen eines Eingangssignals für Gatter mit zwei oder mehr Eingängen als "0" gewertet wird. Die Festlegung der Behandlung des "u"-Werts erfolgte willkürlich von den Entwicklern von VSIM und stellt Determinismus bei der Abarbeitung sicher. Durch Ignorieren des Takts geht die eigentliche Funktionalität des Flipflops und seine Aufgabe als Speicherglied verloren. Als Folge daraus können ggf. Schaltungsteile, die nur von FF-Ausgängen getrieben werden, während der gesamten Simulation inaktiv bleiben. Dies zeigt sich in einigen wenigen, auch für beliebige Stimulilängen konstant auf dem logischen Wert "u" verbleibenden Ausgangssignalen[11]. Jedoch erscheint die Anzahl der simulierten Ereignisse signifikant genug, um diese Vereinfachung zu rechtfertigen, zumal das konkrete Verhalten der Schaltungen nicht bekannt ist und es somit nicht direkt um Korrektheitskontrollen der Schaltfunktion geht.

Der Wert des in der zweiten Stufe des FF verwendeten negierten Clock-Signals[12] ist bei dieser Vorgehensweise ebenfalls undefiniert. Es wird als "0" behandelt, so daß die davon getriebenen

[10] der Defaultwert ist 100 ns bei einer verwendeten Schaltzeit von 2 ns pro Gatter und 1 ns pro Inverter

[11] d.h., es erfolgte kein Signalwechsel über die gesamte Simulationsdauer.

[12] VSIM modelliert einen Inverter so, daß dieser nur bei Anliegen einer logischen Null den Wert "1" generiert; ansonsten bleibt der Wert auf "0". In unserem Fall wird sogar "u" beibehalten, da sich der Eingang überhaupt nicht ändert.

NAND-Gatter stets in Abhängigkeit von den Dateneingängen schalten und somit eine höhere Ereignisrate erzielt werden kann. Ein ähnliches Verhalten tritt auf, wenn der Takt permanent den Wert "1" besitzt. Dann vervielfacht sich die Ereignisrate jedoch nochmals, da die erste Latchstufe alle Eingangsänderungen ihres Datenkanals verarbeitet, auch wenn die 2. Stufe keine Werte übernimmt. Wird dagegen Clock mit "0" belegt, so schalten weder die 1. noch die 2. Stufe Signaländerungen durch, und es fallen nur wenige Ereignisse an. Bei zufällig gewähltem Clock-Takt ist die Simulation ebenfalls sehr schnell beendet, da die zweite Latchstufe nur bei Wechsel des Clock-Signals Daten übernimmt.

7.3 Sequentielle Simulationsergebnisse

Die folgenden Messungen erfolgten mit einer optimierten Version des rein sequentiellen Simulators VSIM auf einem einizigen Knoten des GC/PP und stellen die Grundlage aller nachfolgenden Beurteilungen dar. Bereits bei sehr kurzen Läufen (ein einziger Signalwechsel der Primäreingänge oder entsprechend 100 ns Simulationslänge) zeigten sich an den Ausgängen Zustandsänderungen. Die gewählte Vektorlänge beträgt für die hier vorgestellten Untersuchungen 64 für die ISCAS-Schaltungen und 1024 für die künstlichen Benchmarks (entsprechend einer simulierten Zeitdauer von 6,4 bzw. 102,4 μs). Im folgenden werden zunächst die Ergebnisse des Grundverfahrens genannt, bevor der Einfluß der zufälligen Stimuli und der Granularität auf die Messungen diskutiert werden.

Modell	Simulationsdauer	Standardabweichung	Ereignisse	Ereignisse/Sekunde
s1196	1,716	0,006	132176	77036
s1196dff	21,896	0,016	494298	22574
s13207	64,446	0,070	3780728	58664
s13207dff	1116,340	0,259	20525375	18385
s13207dff2	7,476	0,001	69965	9358
s35932	211,882	0,191	11944722	56373
s35932dff	2811,234	0,003	65966512	23465
s35932dff2	29,203	0,003	762586	26112
inv100000_chain	789,791	0,882	52451051	66410
inv10000_8_branch	132,283	0,000	10081226	76209
inv10000_8_cs	1803,047	0,000	138619353	76880
inv10000_8_inout	133,401	0,000	10064498	75445
inv10000_chain	128,225	0,087	9746951	76013
inv10000_cycle	55,780	0,169	4718773	84597
inv200000_chain	795,614	0,001	52481026	65962
inv64_8_branch	0,764	0,000	73730	96452
inv64_8_cs	8,131	0,000	819979	100847
inv64_8_inout	0,867	0,000	72706	83897
inv64_chain	1,135	0,004	67586	59541

Tabelle 7.2: Reine sequentielle Simulationsdauer (in Sek.)

7.3.1 Grundverfahren

Tabelle 7.2 zeigt die reine Simulationsdauer[13] t_{sim} und die Anzahl der simulierten Ereignisse für die oben genannten Eingabedaten. Die sequentiellen Simulationen zeigten sich als äußerst stabil bezüglich der Laufzeit. Es gab keine wesentlichen Abweichung vom errechneten Mittelwert auf dem GC/PP-Rechner. Die Meßwerte bewegten sich maximal 1 Prozent um diesen Wert.

Mit dem sequentiellen Simulator kann also eine nicht unerhebliche Ereignisrate erzielt werden[14]. Neben der eigentlichen Simulationsdauer entstehen jedoch, wie in Kapitel 3.2 beschrieben, weitere Kosten. Bei der sequentiellen Simulation fallen nur die Zeiten für das Einlesen des Modells t_{input} und für das Aufbereiten und die Ausgabe der Ergebnisse t_{output} an. Im konkreten Fall ist sogar t_{output} nahezu Null und wird deshalb vernachlässigt. Tabelle 7.3 zeigt für die untersuchten Modelle die beiden relevanten Einzelwerte und die akkumulierte Gesamtdauer. Partitionierung, Verteilung des Modells sowie Einsammeln der Ergebnisse entfallen natürlicherweise.

Die Werte, die auf einer Sun SparcStation 20 gemessen wurden, fallen für einzelne Modelle etwas höher aus als auf dem GC/PP. Außerdem treten stärkere Schwankungen in den Laufzeiten auf, die jedoch durch die Belastung des Rechners durch andere Benutzer hervorgerufen werden[15]. Sie bewegen sich im Bereich bis zu ± 20 Prozent um den Mittelwert. Der Dateizugriff auf die Modellbeschreibung erfolgt dagegen mehr als doppelt so schnell wie auf dem GC/PP. Insgesamt heben die schnelleren Dateizugriffe langsamere Laufzeiten jedoch nicht auf.

Modell	t_{input}	t_{sim}	Σ
GC/PP			
s1196	0,339	1,716	2,055
s13207	5,471	64,449	69,920
s35932	17,771	211,882	229,653
inv64_chain	0,083	1,135	1,218
inv10000_chain	3,341	128,225	131,566
inv100000_chain	36,321	789,791	826,112
Workstation (Sun SparcStation20)			
s1196	0,133	2,194	2,327
s13207	2,382	63,989	66,371
s35932	7,638	267,289	274,927
inv64_chain	0,014	1,121	1,135
inv10000_chain	1,383	135,439	136,822
inv100000_chain	16,506	828,024	844,530

Tabelle 7.3: Gesamtzeitbedarf bei sequentieller Simulation (in Sek.)

[13] Alle Angaben sind für die realen Schaltungen in der Regel über fünf oder mehr unabhängige Simulationsläufe gemittelt.

[14] Werte kommerzieller Produkte wie Synopsys, Mentor, Vantage waren nicht verfügbar, so daß eine vergleichende Einordnung leider nicht möglich ist. Auf Nachfrage bei Cadence wurde unsere Schaltung inv10000_chain mit deren Simulator "Affirma NC Verilog Simulator" auf einer SUN Ultra-2 mit 200 MHz in 64 CPU-Sekunden simuliert. Vergleichsmessungen mit dem sequentiellen Simulator VSIM ergaben Berechnungsdauern zwischen 37 und 42 CPU-Sekunden auf einer Ultra-2 mit 300 MHz.

[15] Die Messungen erfolgten zu normalen Zeiten mit allgemeinem Rechenbetrieb.

7.3.2 Variation der Eingaben

Zur Prüfung der Sensibilität der Modelle gegenüber variierenden Eingaben wurden Testläufe mit 10 Sätzen verschiedener Eingabevektoren von 6,4 μs Simulationsdauer durchgeführt. Aufgrund ihres zufälligen Erzeugungscharakters haben sie jedoch einen stochastisch ähnlichen Aufbau. Die Änderungswahrscheinlichkeit zwischen zwei aufeinanderfolgenden Werten beträgt stets 50 Prozent.

Tabelle 7.4 stellt die Mittelwerte und Standardabweichung σ aus Tabelle 7.2 den Messungen mit variierenden Eingabevektoren gegenüber. Es zeigt sich, daß bei den realen sequentiellen Schaltungen keine Sensibilität gegenüber den Eingaben vorzuliegen scheint. Auch bei den extremen Ansätzen mit konstanten Werten "0" bzw. "1" für das Clock-Signal zeigte sich eine vergleichbare Ereignisrate pro Sekunde wie bei zufallsgenerierten Stimuli.

	1 Eingabevektor			10 Eingabevektoren		
Modell	t_{sim}	Events/Sek.	σ	t_{sim}	Events/Sek.	σ
s1196	1,716	77.036	0,006	1,731	76.859	0,017
s13207	64,449	58.664	0,070	64,495	58.626	0,119
s35932	211,882	56.373	0,191	208,645	56.462	1,524

Tabelle 7.4: Einfluß der Eingabevektoren

7.3.3 Unterschiedliche Granularität

Die bisherigen Messungen wurden mit einer sehr niedrigen Ereignisgranularität durchgeführt. Deshalb werden nachfolgend die Meßwerte für die identischen Berechnungen mit erhöhter Granularität durchgeführt, da die Granularität insbesondere im verteilten Simulationsfall zusammen mit der Nachrichtenlaufzeit von Bedeutung ist.

Modell	Simulationsdauer	Ereignisse	Ereignisse/Sekunde
s1196	1,815	132.176	72.832
s13207	67,508	3.780.728	56.003
s35932	220,907	11.944.722	54.070
inv10000_chain	133,255	9.746.951	73.145

Tabelle 7.5: Sequentielle Simulation mit Verzögerung 0

Modell	Simulationsdauer	Ereignisse	Ereignisse/Sekunde
s1196	61	132.176	2.153
s13207	1.707	3.780.728	2.214
s35932	5.392	11.944.722	2.214
inv10000_chain	2.803	9.746.951	3.477
inv100000_chain	15.188	52.451.051	3.453

Tabelle 7.6: Sequentielle Simulation mit Verzögerungsfaktor 30

Die Erhöhung der Granularität erfolgt durch eine Schleife mit leerem Körper, die im Anschluß an die Abarbeitung eines Ereignisses solange durchlaufen wird, bis die voreingestellte Zeit erreicht ist. Prinzipiell sollten die somit erzielten Meßwerte identisch zu den Ergebnissen mit niedriger Granularität multipliziert mit dem Verzögerungsfaktor sein.

Die Länge der Schleife wird bei der Initialisierung des Simulators dynamisch bestimmt, indem der maximale Wert der Zählvariablen ermittelt wird, so daß die Schleifenabarbeitung die entsprechende Verzögerungszeit verbraucht. Die Dauer der zusätzlichen Verzögerung für die Ereignisbearbeitung (in μs) entnimmt der Simulator der Umgebungsvariablen VSIM_EVAL_DELAY[16]. Da der Simulatorcode durch Einführung der Verzögerungskomponente leicht modifiziert ist, wurden ebenfalls Messungen für eine Erhöhung der Granularität um 0 Sekunden als Vergleichswert zur Ermittlung der Meßwertverfälschung durchgeführt. Die Ergebnisse sind dort jedoch weitgehend identisch zum Grundverfahren. Weitere Meßpunkte sind die Verdreißigfachung der Grundgranularität und eine Vervielfachung um den Faktor 150. Dadurch ergeben sich als Verhältnisse zwischen Nachrichtenlaufzeit und Ereignisdauer 1:1 bzw. 1:5.

Modell	Simulationsdauer	Ereignisse	Ereignisse/Sekunde
s1196	316	132.176	417
s13207	8.734	3.780.728	432
s35932	27.278	11.944.722	438
inv10000_chain	14.905	9.746.951	654
inv100000_chain	80.277	52.451.051	653

Tabelle 7.7: Sequentielle Simulation mit Verzögerungsfaktor 150

Die Tabellen 7.5, 7.6 und 7.7 zeigen die Meßergebnisse bei verschiedener Granularität für die verwendeten Faktoren, die bei den parallelen Versionen bezüglich des Verhältnisses des Kommunikations- zum Berechnungsaufwand als Überwiegen der Kommunikationsseite, als Ausgewogenheit zwischen beiden und als starker Überhang der Berechnungsseite interpretiert werden kann. Die Simulationsdauer steigt natürlich und wie erwartet proportional zur Erhöhung der Granularität.

7.4 Kritische-Pfad-Analyse der Schaltkreise

Es stellt sich nun die Frage, ob die betrachteten Modelle überhaupt genügend Potential zur Beschleunigung besitzen. Deshalb wird in diesem Abschnitt unter den idealisierten und unrealistischen Annahmen von Kommunikation ohne Zeitbedarf und unbegrenzten Rechnerressourcen das CPA-Verfahren aus Kapitel 3.3.2 zur Ermittlung der für die parallele Simulation der betrachteten Schaltungen minimal benötigten Zeit eingesetzt. Dabei werden alle parallel ausführbaren Ereignisse virtuell gleichzeitig abgearbeitet, wodurch eine Obergrenze für die mit einem Modell unter Einhaltung der Kausalität erreichbare Beschleunigung anhand des kritischen Pfads definiert ist. Insbesondere die künstlichen Modelle (z.B. lineare Kette von Invertern) erlauben eine einfache Kontrolle der Korrektheit der berechneten Werte.

7.4.1 Basismessungen mit unveränderter Granularität

Tabelle 7.8 gibt die benötigte Realzeit[17] zur Berechnung des kritischen Pfads (CPA-Laufzeit), seine Länge (unter der Annahme, daß Kommunikation nichts kostet), den Beschleunigungsfaktor als Quotient dieser beiden Werte und die in der 5. Tabellenspalte angegebene reine sequentielle Simulationsdauer als Summe der reinen Ereignisbearbeitungszeiten wieder. Dieser letzte Wert berechnet

[16] Die Steuerung der als simulationsrelvant betrachteten Parameter über die eingeführten Umgebungsvariablen ermöglicht eine einfache automatische Durchführung verschiedenen Meßreihen für VSIM und DVSIM.

[17] Alle Laufzeiten werden in Sekunden angegeben.

sich aus der gemessenen CPA-Laufzeit der zweiten Spalte abzüglich 10 μs pro Ereignis als Kompensationswert für die Zeitmessung[18]. Die errechneten Werte stimmen mit den im Basisverfahren (vgl. Tabelle 7.2) ermittelten sequentiellen Laufzeiten nahezu überein und zeigen die notwendige Exaktheit bei der Messung der Ereignisbearbeitungszeiten. Der Beschleunigungsfaktor stellt die potentiell erreichbare Beschleunigung dar, die für die gewählten Eingabewerte durch parallele Simulation erreicht werden kann[19].

Die beobachteten Werte zeigen an, daß in den verwendeten Benchmarks prinzipiell ein Potential zur Parallelisierung existiert. Erstaunlich ist, daß die Länge der Berechnung auf dem kritischen Pfad in allen Fällen deutlich unterhalb einer halben Sekunde liegt.

Modell	CPA-Laufzeit	Länge kritischer Pfad	Beschleunigung	seq. Simulationsdauer
s1196	2,967	0,047	64	1,645
s13207	102,226	0,198	517	64,419
s35932	344,373	0,138	2.472	224,926

Tabelle 7.8: Inhärentes Beschleunigungspotential bei unbegrenzten Ressourcen

Zur Validierung der Korrektheit der Meßverfahren wurden insbesondere die synthetischen Benchmarks, die aus einer einzigen Kette von Invertern bestehen (inv$n$_chain), untersucht. Dort ist ein nahezu linearer Speedup zu erwarten, da sie einen pipelineähnlichen Aufbau besitzen und hohen Parallelitätsgrad versprechen. Die tatsächlichen Meßergebnisse zeigen allerdings ein anderes Verhalten. Verwendet man die für die realen Modelle bereits zur Produktion verläßlicher Ergebnisse ausreichende Simulationsdauer von 6,4 μs, so werden gerade 50 Prozent des maximalen Speedups beobachtet.

Ein naheliegender Gedanke ist, daß sich die Pipeline der gleichzeitig auswertbaren Objekte erst einmal in einer Anlaufphase mit Ereignissen füllen muß. Deshalb wurde die Länge der Eingabevektoren schrittweise verdoppelt, um das Verhalten für längere Simulationsläufe zu überprüfen (Tabelle 7.9). Nachdem die Pipeline gefüllt ist, erhöhen sich mit zunehmender Simulationslänge auch die erreichbaren Beschleunigungswerte. Es ergaben sich bei der Auswertung jedoch zum Teil widersprüchliche Phänomene: Bei geringer Objektanzahl erreicht die Beschleunigung ein superlineares Wachstum; bei den größeren Modellen bleiben die erreichbaren Werte jedoch unerwartet niedrig.

Bei der Betrachtung der Einschwingvorgänge fällt auf, daß mit Modell inv64_chain für längere Simulationsläufe ein mit 64 Objekten theoretisch gar nicht möglicher superlinearer Speedup auftritt. Die Ursachen dafür sind vermutlich Meßungenauigkeiten, die bei der geringen Granularität zu einer überzogenen Verfälschung führen. Diese Hypothese wird durch die Ergebnisse bei erhöhter Ereignisbearbeitungszeit gestützt, bei denen sich die Beschleunigung nur noch asymptotisch der Objektanzahl annähert. Die korrespondierenden Werte für gleiche Stimulilängen stimmen bei unterschiedlichen Granularitäten für alle Modelle sehr gut überein.

Für die beiden größeren Modelle inv10000_chain und inv100000_chain wächst weiterhin der Speedup nur relativ langsam im Verhältnis zur Modellgröße. Ursachen dafür sind die Wahl der Stimulilänge im Zusammenspiel mit der virtuellen Verfügbarkeit einer unbegrenzten Prozessoranzahl. Die Folge der durch die Stimuli definierten Ereignisse schwappt als Welle durch die lineare

[18] Systematische Meßfehler, die durch die zusätzliche Zeitenermittlung zur Berechnung des kritischen Pfads entstehen, wurden hier nicht von vornherein herausgerechnet, um die Korrektheit der Kompenstionsmethode (bzw. die Exaktheit bei der Abschätzung der Systemaufrufe zur Zeitmessung) zu überprüfen.

[19] Bei Integration eines Korrekturwerts für die Aufrufe der Zeitmeßfunktionen fallen die erreichbaren Beschleunigungen ca. 10 Prozent niedriger aus. Das gilt jedoch nur für den Fall unveränderter Granularität. Bei Erhöhung der Ereignisgranularität ist der Korrekturfaktor ohnehin verschwindend gering.

Kette von Objekt zu Objekt. Jedes Objekt arbeitet somit in einer sehr kurzen Zeit alle für es relevanten Ereignisse ab. Bei einer Länge n der Stimulidaten überlappen sich maximal n gleichzeitige Ereignisausführungen. Somit begrenzt die Länge der Eingabevektoren bzw. die Anzahl der zu einem Realzeitpunkt existenten Ereignisse ähnlich wie bei der zeitgesteuerten Simulation auch die rein theoretisch erreichbare maximale Beschleunigung auf einen Wert, der wesentlich unterhalb eines linearen Speedups bezogen auf die Objektanzahl liegen kann. Die mit "WS" gekennzeichnete Messung erfolgte auf einer Workstation mit einer sehr langen Eingabe[20] und bestätigt in beschränktem Rahmen die Beobachtungen, daß die Stimulilänge die Parallelität beschränken kann, wenn das Modell eine konstante Ereigniskomplexität aufweist, d.h. jedes Ereignis genau ein Folgeereignis generiert.

Modell	Stimulivektor-länge	Beschleunigung bei Granularität		
		1	30	150
inv64_chain	64	33,961	32,620	32,317
inv64_chain	128	49,341	43,412	43,007
inv64_chain	256	61,113	52,242	51,505
inv64_chain	512	79,357	58,037	57,200
inv64_chain	1024	88,696	61,470	60,522
inv10000_chain	64	24,957	31,843	32,376
inv10000_chain	128	50,710	75,421	77,951
inv10000_chain	256	203,296	198,998	203,293
inv10000_chain	512	307,233	444,282	444,216
inv10000_chain	1024	661,449	893,366	892,454
inv100000_chain	64	24,919	31,849	32,370
inv100000_chain	128	38,582	61,257	63,863
inv100000_chain	256	70,197	120,411	126,837
inv100000_chain	512	133,346	238,925	252,841
inv100000_chain	1024	265,428	485,818	516,462
WS inv100000_chain	16384	4121 – 8137		

Tabelle 7.9: Einfluß von Einschwingvorgängen

Bei Beobachtungen mit der Basisgranularität zur Gatterauswertung werden bei den größeren Modellen aber auch noch geringere Beschleunigungen beobachtet als die durch die Stimulilänge vorgegebenen. Diese sind durch den starken Einfluß der Varianzen der parallelen Ereignisabarbeitung bei kurzer Ereignisdauer begründet. Verschieben sich die realen Ausführungszeiten zweier paralleler Ereignisse z.B. aufgrund von Lastschwankungen oder Meßungenauigkeiten bei niedriger Granularität um einen kleinen absoluten Betrag, so hat dies einen wesentlich stärkeren Einfluß auf die Länge des kritischen Pfads als wenn dieselbe Abweichung bei einer längeren Ereignisbearbeitungsdauer auftritt.

Die in Abbildung 7.2a gezeigte exakte Parallelausführung kommt in der Realität so gut wie nie vor, da normalerweise stets leichte Schwankungen der Ereignisroutinenlängen und des Beginns der realen Abarbeitung eines Ereignisses auch bei synchronem Start für zwei Prozessoren auftreten (Abbildung 7.2b). Diese Differenz ist bei feiner Granularität, wie erwähnt, gravierender als bei

[20] Simuliert wurden knapp 1,6 Milliarden Ereignisse für einen Simulationszeitraum von 16,384 ms; der Simulator lief dazu über 23 Stunden. Um die Überläufe der Realzeituhren zu vermeiden wurde die sequentielle Version auf einer Workstation verwendet. Die Berechnungsdauer für die CPA war nahezu identisch zur GC/PP-Version, jedoch lieferte die WS-Version stark streuende Beschleunigungsvorhersagen.

grober. In Abb. 7.2b verlängert sich somit die Ausführungszeit der beiden parallelen Ereignisse um ca. 20 Prozent, während in 7.2c bei gleicher Verschiebung lediglich 7 Prozent mehr an Rechenzeit benötigt wird.

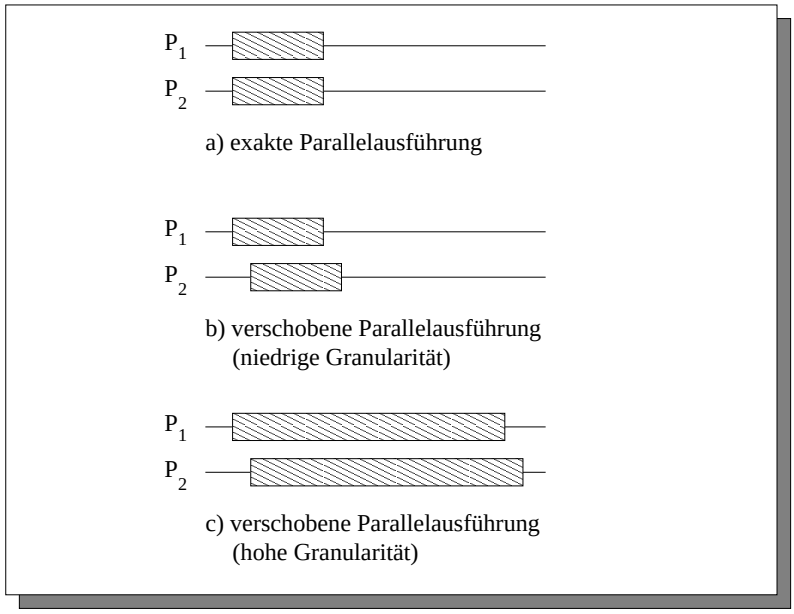


Abbildung 7.2: Einfluß der Meßgenauigkeit auf den kritischen Pfad

7.4.2 Erhöhung der Granularität

Erwartungsgemäß hat die Erhöhung der Ereignisgranularität keine gravierenden Auswirkungen auf die Länge des kritischen Pfads und den Speedup, da sich die Gesamtdauer der sequentiellen Simulation dadurch ebenfalls proportional erhöht. Tabelle 7.10 zeigt die entsprechenden Ergebnisse für die in Abschnitt 7.3.3 eingeführten Verzögerungen.

Modell	Beschleunigung bei Granularität		
	1	30	150
s1196	63,9	63,7	63,4
s13207	517	847	878
s35932	2.472	2.766	2.786

Tabelle 7.10: Zusammenhang zwischen Granularität und CPA

Für die beiden größeren Modelle treten hier aus den im vorherigen Abschnitt genannten Gründen ebenfalls wieder die Abweichungen der Werte nach unten für Untersuchungen bei niedriger Granularität auf. Da man für die Berechnung der CPA nicht die verlängerten Gesamtlaufzeiten messen muß, sondern der kritische Pfad implizit bei der sequentiellen Simulation berechnet wird, kann die Erhöhung der Granularität durch simple Addition der zusätzlichen Verzögerung auf die Generierungszeitpunkte der Ereignisse modelliert werden. Dadurch vermeidet man die Warteschleifen und erreicht eine wesentlich kürzere Berechnungsdauer mit demselben Aufwand wie bei der Grundgranularität. Eine Abschätzung der Leistung für höhere Granularität ist somit in wesentlich kürzerer

Laufzeit möglich. Der Verzögerungswert wird der Umgebungsvariablen CPA_EVAL_DELAY entnommen.

Die schnelle Beantwortung der Frage, ob sich die parallele Simulation eines Modells lohnt, ist somit mittels Kritische-Pfad-Analyse möglich. Dieses Berechnungsverfahren kann jedoch nur bei Modellen mit niedriger Granularität oder zur effizienten und schnellen Approximation eines äquivalenten Modellverhaltens mit hoher Granularität eingesetzt werden. Für Modelle, bei denen die Ereignisdauer bereits lang ist, erfordert die Berechnung des kritischen Pfads einen entsprechend größeren Zeitraum.

7.4.3 Einfluß der Kommunikation

Der Einfluß der Kommunikationsdauer auf das Laufzeitverhalten wurde von Carothers, Fujimoto und England an einem tatsächlich parallel arbeitenden Time Warp-Simulator untersucht [CFE94a]. Unterschiedliche Dauer der Kommunikation wurde durch einen Leerschleife vor dem Nachrichtenversand auf den benutzten Workstations emuliert. Der Einfluß des Delays macht sich in diesen Betrachtungen erst dann negativ in den Beschleunigungswerten bemerkbar, wenn die Nachrichtenverzögerung (initial werden 3 - 5 ms benötigt) die Größenordnung der Ereignisgranularität (25 ms) erreicht oder übertrifft. Ab einer Kommunikationsdauer von 25 ms brechen somit die Beschleunigungswerte ein.

Bei einer feingranularen Applikation (1 μs / Ereignis) zeigt sich bei einer Erhöhung der künstlichen Verzögerung der Kommunikationszeiten in Schritten von 4 ms ein drastischerer Effizienzverlust als im ersten Beispiel von jeweils 30 - 40 Prozent pro Erhöhung. Die Logiksimulation auf Gatterebene mit einer 30-mal schnelleren Ereignisabarbeitung im Vergleich zu den Nachrichtenlaufzeiten läßt sich in die Klasse feingranularer Applikationen einstufen. Die Verzögerung durch Kommunikationsoperationen hat somit im verteilten Fall großen Einfluß auf die Leistungsfähigkeit, weshalb dieser Parameter auch im Kontext der CPA für die untersuchten Modelle nicht ignoriert werden darf.

Für den GC/PP mit seinen Kommunikationskoprozessoren (Transputer) ist nicht die gesamte Dauer der Nachrichtenübertragung, sondern im wesentlichen nur die Setup-Zeit zur Initiierung der Kommunikation relevant (Initialisierung der Sendepuffer und der DMA-Operationen). Durch die Überlappung der Berechnungen von Hauptprozessor (PPC) und Transputer erfolgt quasi eine Parallelabarbeitung von Kommunikation und Simulation. Die Erhöhung der Granularität des Nachrichtenversands entspricht hier der Verlängerung der Startup-Zeiten, die z.B. durch Marshalling von Paketen in PVM verursacht werden. Beide Sichten, erhöhte Kommunikationsdauer bzw. vermehrter Aufwand für Setup, können aber als äquivalent betrachtet werden, da beide eine verspätete Auslieferung der Nachrichten bewirken. Im ersten Fall wird jedoch lediglich der Empfänger verzögert, während bei erhöhten Setupkosten auch der Sender keine "sinnvolle" Arbeit zur Erledigung der eigentlichen Aufgabe leisten kann.

Im Gegensatz zum Ansatz in [CFE94a] kann beim sequentiellen CPA-Simulator auf eine Warteschleife, die die künstliche Verzögerung des Nachrichtenversands wie bei der Erhöhung der Ereignisgranularität realisiert, verzichtet werden. Zur Modellierung der Kommunikationsverzögerung wird hier lediglich ein zusätzlicher konstanter Wert zum berechneten Erzeugungszeitpunkt aller Ereignisse, die für einen anderen Prozessor als den generierenden bestimmt sind, addiert. Der Wert wird der Umgebungsvariablen CPA_MSG_DELAY entnommen (in μs). Leichte Varianzen der Kommunikationsdauer haben nur einen unbedeutenden Einfluß auf das Verhalten [CFE94a], so daß hier keine zusätzlichen Messungen mit um einen Mittelwert streuenden, zufallsgenerierten Verzögerungen erfolgten. Die Parameterwerte wurden an die Granularitätsmessungen beim sequentiellen

Simulator mit den Verhältnissen von "Ereignisaufwand zu Nachrichtenaufwand" von 1:1 und 1:30 unter Beibehalten einer konstanten Ereignisgranularität von 300 μs angepaßt[21].

Die beiden Varianten entsprechen somit einem günstigen realen Kommunikationsmedium (1:1) und einer teuren Kommunikationsschicht (1:30), wie sie auf dem GC/PP existiert. Eine zusätzliche Betrachtung einer höheren Ereignis- als Nachrichtengranularität ist nicht nötig, da die gemessenen Werte dann nahe bei den idealen Ergebnissen ohne Berücksichtigung der Kommunikation liegen.

Es zeigt sich (Tabelle 7.11), daß die Berücksichtigung der Kommunikationsdauer in der Kritische-Pfad-Analyse (erwarteterweise) drastische Einbrüche im Beschleunigungspotential mit sich bringt. Das gilt insbesondere, wenn der Kommunikationsanteil überhand gewinnt. Feingranulare Applikationen wie Logiksimulation auf Gatterebene weisen deshalb zwar ein interessantes, aber bei weitem kein überwältigendes Potential mehr auf. In den hier betrachteten ISCAS'89-Schaltungen sinkt der erreichbare Speedup auf weniger als 6 Prozent verglichen mit der reinen CPA ohne Beachtung der Kommunikation. Bei einem Gleichgewicht der kritischen Komponenten werden immerhin noch 65 Prozent des ursprünglichen Werts erreicht.

	Ereignisgranularität : Kommunikationsdauer	
Modell	1:1	1:30
s13207	562	52
s35932	1.815	165

Tabelle 7.11: Beschleunigung mit Kommunikationsaspekten bei CPA

Aufgrund der fest vorgegebenen Kommunikationsdauer von 300 μs auf dem GC/PP stellt sich nun die Frage, ob sich das Verhältnis der Granularitäten für einen Endbenutzer sinnvoll d.h. ohne künstliche Verzögerung der Ereignisroutinen verbessern läßt.

7.4.4 Einfluß unterschiedlich granularer Objekte

Eine Möglichkeit zur Erhöhung der Granularität stellen die hierarchischen Modellierungsmöglichkeiten von VHDL dar. Durch Verhaltensbeschreibungen (behavioral view) mittels VHDL-Prozessen lassen sich beliebig komplexe Objekte gestalten. Ein VHDL-Prozeß stellt somit eine Einheit möglicherweise gröberer Granularität dar und wird als ein einziges Objekt behandelt. Alle innerhalb dieses Objekts benötigten Signale werden bei der Partitionierung demselben Prozessor zugewiesen. In den vorliegenden realen Schaltungen wurde die Modellierung der D-Flipflops deshalb zur Granularitätserhöhung in einen VHDL-Prozeß gepackt, indem die verwendeten Gatter einfach mit einem entsprechenden `Process`-Statement geklammert wurden.

Die Ausführung einer Ereignisroutine bewirkt somit die gleichzeitige Auswertung der internen Gatter und ein Aktualisieren der sich dabei ändernden Signalwerte innerhalb eines einzigen Funktionsaufrufs. Die Dauer der Auswertung steigt durch die Einbindung zusätzlichen C-Codes von 10 μs auf 330 μs, also grob um den Faktor 30. Da jedoch eine Großzahl der in den Schaltungen vorhandenen Gatter nicht geclustert werden kann und sie in ihrer Grundform erhalten bleiben, ergeben sich mittlere Ereignisgranularitäten von 50 μs bei Modell s13207dff[22] und 70 μs bei s35932dff.

Die gemittelten Werte alleine sind jedoch nicht sehr aussagekräftig aufgrund der hohen Granularität der der reinen FF-Auswertung. Um die Vorteile der Parallelbearbeitung voll auszunutzen,

[21] Die höhere Bearbeitungsdauer der Ereignisse wurde gewählt, um Verfälschungen durch Meßungenauigkeiten und Probleme mit der zeitgleichen Bearbeitung paralleler Ereignisse, die in den Kapiteln 7.4.1 bzw. 7.4.2 beobachtet wurden, zu vermeiden.

[22] Die Ergänzung `dff` wie im vorliegenden Fall und später auch `dff2` gibt an, welche Implementierung für die Flipflops verwandt wurde.

müssen 30 simple Gatterauswertungen gleichzeitig zu einer einzigen FF-Neuberechnung ausgeführt werden, damit sich der Vorteil der teilweise gröberen Granularität rechnet. Tabelle 7.12 zeigt, daß durch die variierenden Bearbeitungszeiten der maximale Speedup auch bei idealisierter Kommunikation ohne Zeitaufwand für Modell s13207dff zurückgeht. Die zweite Schaltung scheint jedoch invariant gegenüber der veränderten Granularität zu sein und weist ein identisches Verhalten auf. Zusätzlich steigt jedoch auch die Anzahl der generierten Ereignisse sprunghaft an, da die Signale innerhalb der VHDL-Prozesse öfter ausgewertet werden. Die Modellierung von Flipflops durch simple Integration in ein `Process`-Statement ist somit recht ineffizient.

Modell	Beschleunigung
s13207dff	303
s13207dff2	492
s35932dff	2792
s35932dff2	1824

Tabelle 7.12: CPA bei variierender Ereignisgranularität

Da sich außerdem keine Verbesserung bezüglich der Beschleunigung abzeichnet, wurde in einer zweiten Variante (`dff2`) eine optimierte Architektur der Flipflops verwendet. Die Datenübernahme wird einfach durch die fallende Flanke des Clock-Signals getriggert. Dies geschieht mittels elementarer VHDL-Funktionalität (Signalattribut `'falling`), die in einer einfachen `if`-Abfrage verwendet werden kann, um das Eintreten der Bedingung zu testen. Dadurch reduziert sich die Auswertungsdauer auf 110 μs pro FF-Ereignis. Duch das vollständige Entfallen der 10 Gatter pro FF sinkt die Ereignisrate gleichzeitig auf 2 Prozent verglichen mit dem ursprünglichen Modell. Um dennoch Simulationsläufe mit einer höheren Ereignisanzahl bewerten zu können, wurde die Simulationsdauer erhöht. Die mittlere Ereignisgranularität bei s13207dff2 lag erstaunlicherweise recht hoch bei 110 μs, was auf eine hohe Aktivitätsrate der FFs deutet. s35932dff2 erreichte lediglich einen mittleren Wert von 40 μs. Die Speedups mit der modifizierten FF-Struktur liegen bei s13207dff2 nur leicht unterhalb derer, die mit homogener Objektstruktur theoretisch erreicht werden können. Bei s35932dff2 geht etwa ein Drittel der Leistung durch gemischte Granularitäten verloren.

	Ereignisgranularität : Kommunikationsdauer	
Modell	1:1	1:30
s13207dff2	257	18
s35932dff2	2891	315

Tabelle 7.13: CPA bei variierender Ereignisgranularität und mit Kommunikationsaufwand

Berücksichtigt man weiterhin den Einfluß der Kommunikation bei der Ermittlung des kritischen Pfads, so ergeben sich die in Tabelle 7.13 dargestellten Resultate. Während sich Modell s13207dff2 durch die Hinzunahme von Objekten groberer Granularität im Verhältnis zum Basisverfahren wiederum verschlechtert, sind die Einbrüche, die bei s35932dff2 beobachtet werden, geringer als bei der Grundberechnung des kritischen Pfads. Bei Verwendung einer unbegrenzten Anzahl von Simulatoren ist selbst bei dem bereits recht großen Modell s13207 unter Verwendung der für ein lokales Netz realistischen Kommunikationszeiten von 3 ms ein maximaler Speedup von 18 erreichbar.

7.4.5 Zusammenfassung

In diesem Kapitel wurde ein Verfahren angegeben, das anhand der Analyse der Länge des kritischen Pfads und unter der Annahme unlimitierter Prozessoren für feingranulare Applikationen, die Leistungsfähigkeit und das Parallelisierungspotential unter idealisierten und realen Kommunikationsbedingungen abzuschätzen erlaubt. Durch eine einfache Integration in einen vorhanden sequentiellen Simulator lassen sich somit Aussagen treffen, ob ein gegebenes Modell prinzipiell über Beschleunigungspotential verfügt und wie hoch die zu erwartenden Werte sind.

Für grobgranulare Anwendungen, die sich durch eine feingranulare Modellierung approximieren lassen, ist eine Abschätzung des Parallelisierungspotentials in der wesentlich kürzeren Laufzeit des feingranularen Simulators möglich. Voraussetzung ist lediglich, daß dieselben Ereignisreihenfolgen identisch erzeugt werden. So können z.B. komplexe statistische Berechnungen, die keinen Einfluß auf den Simulationsablauf haben (Warteschlangenmodelle), durch die Addition des Realzeitbetrags, der dafür benötigt wird, bei der Berechnung des kritischen Pfades ersetzt werden. Dasselbe gilt für die Kommunikationszeiten, die ebenfalls zum Generierungszeitpunkt der Ereignisse hinzuaddiert werden, ohne explizite Warteschleifen in den Programmcode einfügen zu müssen.

Allerdings hat sich gezeigt, daß eine generelle Aussage zur Parallelisierbarkeit für ganze Applikationsklassen inhärent schwierig ist. Bereits anhand zweier Beispiele (s13207 und s35932) zeigten sich modellabhängig völlig unterschiedliche Verhalten. Selbst bei Betrachtung nur eines einzigen Modells bleiben dem Entwickler des Simulationsmodells sehr viele Freiheiten (z.B. bei der Realisierung einzelner Schaltungsteile wie Flipflops), die für verschiedene Realisierungen von Objekten unterschiedliche Verhaltensweisen zeigen. Im folgenden Kapitel wird die konkrete Leistungsfähigkeit paralleler Simulationsverfahren untersucht. Diese Ergebnisse werden auf Basis der sequentiellen Laufzeiten auf Übereinstimmung mit den Vorhersagen durch die CPA überprüft. Außerdem wird bei den CPA-Experimenten auf die Idealisierung bezüglich der Prozessoranzahlen verzichtet und der kritische Pfad mit vorgegebenen Partitionierungen der Modelle untersucht.

Kapitel 8

Bewertung paralleler Verfahren

Die mittels sequentieller und theoretischer Verfahren gefundenen Ergebnisse bilden den Ausgangspunkt der folgenden Untersuchungen für parallele Ansätze durch die DVSIM-Evaluierungsumgebung. Die aus den sequentiellen Abläufen berechneten Ergebnisse werden den Messungen mit verschiedenen realen Simulationsmethoden unter Berücksichtigung geeignet gewählter Parameter gegenübergestellt. Außerdem erfolgt ein stichprobenartiger Vergleich mit den Ergebnissen des Oraclelog-Verfahrens.

Steht nicht für jeden LP ein eigener Prozessor zur Verfügung, so sind alle Simulationsläufe von einer konkreten Partitionierung und Zuordnung von LPs an einzelne Prozessoren abhängig. Die Aufteilung der Modelle spielt deshalb im folgenden eine wesentliche Rolle bei der Bewertung (siehe Kap. 5). Zur Beurteilung der Parallelisierbarkeit und der Güte tatsächlich erreichbarer Ergebnisse wurde jeweils ein Vertreter aus den beiden Familien von Synchronisationsprotokollen realisiert und auch CPA-Untersuchungen bei konkreten Partitionierungen untersucht.

Abschließend wird der Aufwand der in Kapitel 3.2 beschriebenen Zusatzkosten bei der parallelen Simulation mit der reinen Simulationsdauer zu den Gesamtkosten verbunden, die letztendlich eine reale Aussage über das Parallelisierungspotential ereignisgesteuerter Simulation gestatten.

8.1 Konservative Verfahren

Als Vertreter der konservativen Verfahren wurde ein Deadlockerkennungs- und -auflösungsverfahren nach Misra [MIS86a] gewählt. Es erkennt globale Deadlocks und die damit äquivalente Terminierung des Systems bezüglich des Blockadezeitpunkts nach dem Doppelzählverfahren (Zwei-Wellen-Algorithmus) von Mattern [MAT89d].

Die ersten Meßergebnisse, die mit einer initialen Version erzielt wurden, waren mehr als enttäuschend. Bei den meisten Modellen ergaben sich Speedup-Werte unter "1" was effektiv einer Verlangsamung der Simulation durch den Einsatz paralleler Techniken entspricht[1]. Nach sorgfältiger Optimierung der Datenstrukturen, die zum Großteil aus dem sequentiellen Code entstammen, konnten bei vielen Modellen moderate Speedups festgestellt werden. In einigen Fällen traten plötzlich auch superlineare Vergleichswerte auf, die jedoch nach der Integration der Optimierungen auch in die sequentiellen Simulatoren wieder auf ein realistisches Maß sanken. Selbstverständlich wurden *alle relevanten* Verbesserungen zum Erhalt der fairen Beobachtung in die sequentiellen Simulator-

[1]Man beachte, daß es keine negativen Speedups gibt. Diese entsprächen aufgrund der Definition nämlich einer Rückwärtsbewegung in der Realzeit. Die entscheidende Marke für Verbesserung und Verlust liegt beim Break-even-Wert "1", an dem sequentieller und paralleler Simulator gleich schnell sind.

versionen von VSIM integriert, die dadurch als Randeffekt ebenfalls wesentlich beschleunigt werden konnten.

8.1.1 Bewertung des Grundalgorithmus

Bei den folgenden Messungen wird die Grundgranularität der Simulationsereignisse und Kommunikationsdauer auf dem GC/PP-Parallelrechner zugrunde gelegt, deren Verhältnis ca. 1:30 beträgt[2].

8.1.1.1 Reale Benchmarks

Die Laufzeiten der Simulationen für die ISCAS'89-Benchmarks sind in Abhängigkeit vom verwendeten Partitionierungsverfahren und der Anzahl der Partitionen (dies entspricht auch der Anzahl der mindestens belegten Prozessoren[3]) in den Tabellen 8.1 – 8.3 dargestellt[4]. Die Abbildungen 8.1 – 8.3 geben die erreichten Beschleunigungswerte bezüglich der reinen Simulationsphasen wieder[5].

Anzahl Part.	Simulationsdauer								
	RR	azykl.	Fan-in Cones		Fan-out Cones		K/L	Soccer	String
	r	a	unbal. c	bal. d	unbal. C	bal. D	k	s	S
1	2,89								
2	61,91	6,14	2,38	2,85	5,53	5,74	30,37	20,46	28,54
4	87,70	2,92	1,75	1,92	3,05	53,62	58,42	38,12	54,36
8	166,87	1,89	2,02	93,98	3,71	106,65	117,59	99,70	106,91
12			3,53	153,88	3,83	164,15			164,73
16	252,66	3,33	3,59	203,46	3,80	221,25	220,04	225,84	219,25
24			3,98	342,16	4,39	364,09			471,13
32			4,24	482,16	10,63	639,28			579,97

Tabelle 8.1: Simulationsdauer mit konservativem Verfahren, s1196

Das kleinste betrachtete Modell mit lediglich 892 Gattern zeigt ein extremes Verhalten. Es gibt nur wenige Experimente, die bereits nach sehr kurzer Zeit beendet sind. Keine Partitionierung erreicht die Geschwindigkeit des sequentiellen Simulators ($t_{sim} \sim$ 1,7 Sekunden). Insbesondere bei Erhöhung der Partitionsanzahlen explodiert die Laufzeit auf ein nicht tragbares Maß. Die die Kommunikationskosten minimierenden Verfahren Soccer und Kernighan-Lin (K/L) weisen bereits bei 2 Partitionen zehnmal schlechtere Laufzeiten auf als das sequentielle Pendant. Dasselbe gilt verstärkt noch für die Verteilung nach dem Round-robin-Verfahren. Ursache ist der überstarke Anteil der Kommunikationskosten, der bei einer Anzahl von etwa 132000 Ereignissen den Löwenanteil ausmacht. Mit zunehmender Partitionsanzahl werden mehr und mehr Nachrichten für den Austausch von Ereignissen benötigt.

[2]Untersuchungen mit diesem realen Verhältnis werden als "Beobachtungen mit Grundgranularität" bezeichnet. Des weiteren werden später auch Untersuchungen mit "ausgewogener Bearbeitungsdauer" (1:1) und "überwiegender Dauer der Ereignisbearbeitung" (5:1) betrachtet.

[3]Auf dem GC/PP wird dem Benutzer stets ein Vielfaches von 4 Doppelprozessorknoten zugewiesen.

[4]Die angegebenen Zeiten entsprechen der Dauer der Simulationsphase $T_{sim,n}$ in Sekunden.

[5]Es wurden nicht immer vollständige Meßreihen für alle Konfigurationen ausgeführt. Ursache waren zum einen die Belegung des Parallelrechners durch andere Benutzer und in seltenen Fällen Programmabbrüche wegen Ressourcenmangel oder von Hand gestoppte Simulationsläufe wegen extremen Laufzeiten. Für diese Fälle finden sich in den Tabellen und Graphen keine Einträge.

Ein noch relativ gutes Verhalten besitzen die strukturorientierten Kegelpartitionierungen, sofern sie nicht größenbalanciert sind. Mit 4 Prozessoren läuft Fan-in-cone nahezu so schnell wie der sequentielle Simulator. Dieses Ergebnis ist nicht unbedingt befriedigend, aber im Vergleich zu den anderen Resultaten doch nennenswert. Ursache für das gute Abschneiden der unbalancierten Versionen ist die Existenz einiger weniger großer Partitionen. Viele Berechnungen für im Simulationsmodell benachbarte Objekte geschehen dadurch lokal auf einem Rechner und nur wenige Nachrichten müssen verschickt werden. Die Simulation ähnelt dadurch eher einer sequentiellen Simulation auf den wenigen Prozessoren mit großen Partitionen, die nur gelegentlich auf Nachrichten anderer Prozessoren warten müssen.

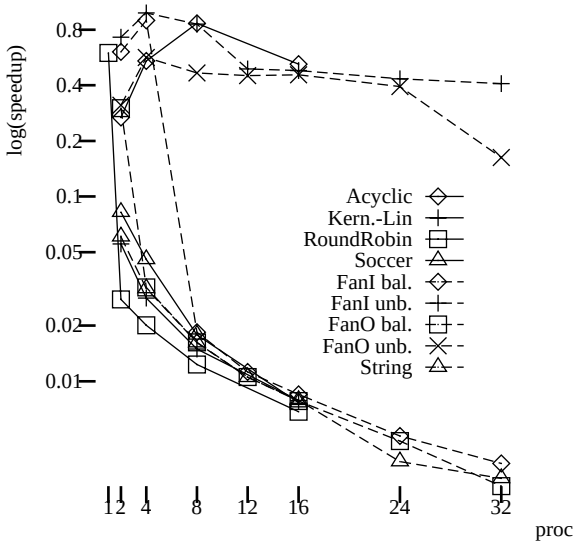


Abbildung 8.1: Speedup, s1196

Balancierte Verfahren tauschen dagegen wesentlich häufiger Nachrichten aus, obwohl auch sie zusammenhängende Schaltungselemente bearbeiten. Hinzukommt, daß in der betrachteten Implementierung der balancierten Kegelpartitionierungen die Berechnung der Kegel in einer Tiefensuche (ähnlich zu String) erfolgt, so daß bei langen Pfaden durch die Schaltung stark entartete Kegel entstehen[6]. K/L, Soccer und Round-robin weisen dieselben Tendenzen bezüglich der Verlangsamung mit zunehmenden Prozessoranzahlen auf. Bei s13207 wurde deshalb nur noch eine Testmessung mit 2 Prozessoren durchgeführt und im folgenden gänzlich auf Round-robin verzichtet.

Abgesehen von den Geschwindigkeitseinbußen ist jedoch festzustellen, daß bei Betrachtung der Beschleunigungen relativ zu den verteilten Simulationsalgorithmen auf nur einem einzigen Prozessor durchaus Beschleunigungen bei einigen Partitionierungen auftreten, deren Werte im Bereich zwischen 2 und 8 Prozessoren auch wachsen. Diese Beobachtung zeigt klar die Notwendigkeit der fairen Bewertung paralleler Simulationszeiten auf Basis der Meßwerte eines wirklich sequentiellen Simulators.

Da das kleinste Modell in keinem Fall einen realen Speedup oberhalb von "1" erreichte, wurden im folgenden keine weiteren Untersuchungen mit Grundgranularität mehr dafür durchgeführt. Die Plausibilität der obigen Erklärungen wird anhand der Beobachtungen bei den beiden größeren Modellen s13207 (15709 Objekte) und s35932 (40695 Objekte) belegt, die ähnliche Verhalten zeigen.

[6]Eine balancierte Kegelpartitionierung nach dem Breitensuch-Algorithmus ließe sich leicht in das Framework integrieren.

Bei den Untersuchungen des Modells s13207 konnten echte kleinere Speedups gegenüber der sequentiellen Variante beobachtet werden. Die Beschleunigungswerte sacken jedoch ab einer Anzahl oberhalb von 4 verwendeten Prozessoren wieder stark ab. Gegenüber der Simulation mit dem parallelen Algorithmus auf genau einem Prozessor sind wieder die bereits beim ersten Benchmark beobachteten überzogenen Beschleunigungen zu vermerken, die aufgrund des hier ca. 30-prozentigen Overheads der verteilten Einprozessorversion gegenüber der sequentiellen Variante eine starke Verzerrung liefern[7].

Anzahl Part.	Simulationsdauer								
	RR	azykl.	Fan-in Cones		Fan-out Cones		K/L	Soccer	String
			unbal.	bal.	unbal.	bal.			
1					94,69				
2	2376,62	54,13	64,92	73,63	55,74	72,83	303,01	80,23	510,82
4		60,77	91,03	69,31	85,63	67,36	183,09	81,22	387,82
8		80,99	87,21	94,73	90,61	107,85	216,98	133,15	338,14
12		84,81	86,77	136,17	90,46	134,38	279,00	197,20	375,70
16		85,32	155,75	167,06	89,97	192,30	360,25	287,93	393,55
24		83,37	76,60	281,86	85,43	272,92	476,84	437,92	503,86
32		83,49	72,94	398,14	88,96	463,38	536,31	520,15	600,19

Tabelle 8.2: Simulationsdauer mit konservativem Verfahren, s13207

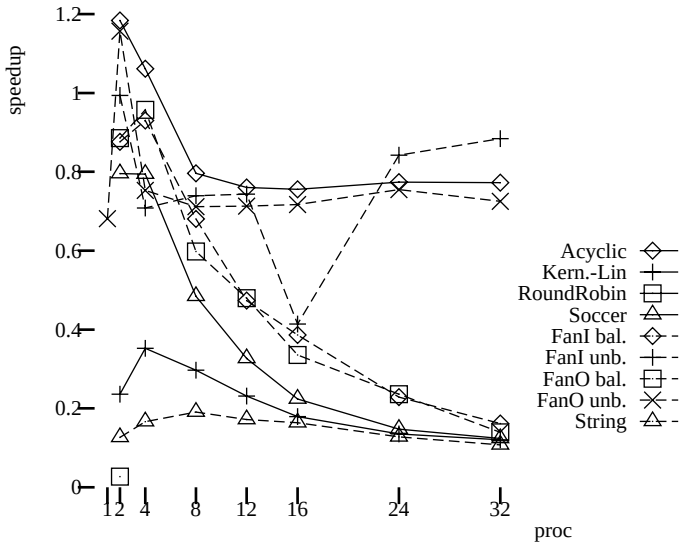


Abbildung 8.2: Speedup, s13207

Die partitionsgrößenbalancierenden Verfahren zeigen eine Verbesserung der Beschleunigung bei der Erhöhung der Partitionszahlen bis zu 4 Knoten. Gegenüber der Simulation auf zwei Knoten gewinnt man also hier mit der Hinzunahme weiterer Prozessoren. Die Ursache dafür liegt jedoch wohl in der recht ineffizienten Bearbeitung und dem hohem Nachrichtenaufkommen bei einer Simulation

[7] Bei den kürzeren Laufzeiten von s1196 fiel der zusätzliche Aufwand mit über 40 Prozent noch stärker ins Gewicht.

auf zwei Rechnern. Ab acht verwendeten Prozessoren bricht die Leistung dieser Methoden wieder stark ein.

Anzahl Part.	Simulationsdauer							
	azykl.	Fan-in Cones		Fan-out Cones		K/L	Soccer	String
		unbal.	bal.	unbal.	bal.			
1						316,06		306,63
2	283,82	306,57	191,74	284,31	286,87	843,07	246,12	
4	156,85	306,71	138,04	164,25	157,50	585,86	139,80	
8	112,51	306,37	138,54	104,04	101,66	465,57	171,89	2211,55
12	52,05	305,96	204,78	59,09	135,03	462,95	185,18	2369,40
16	60,03	307,63	206,33	59,40	186,12	412,84	245,51	1702,49
24	52,01	306,69	275,88	77,65	263,25	460,84	273,95	1320,73
32	84,28	307,34	345,01	85,70	334,74	529,44	429,37	1007,46

Tabelle 8.3: Simulationsdauer mit konservativem Verfahren, s35932

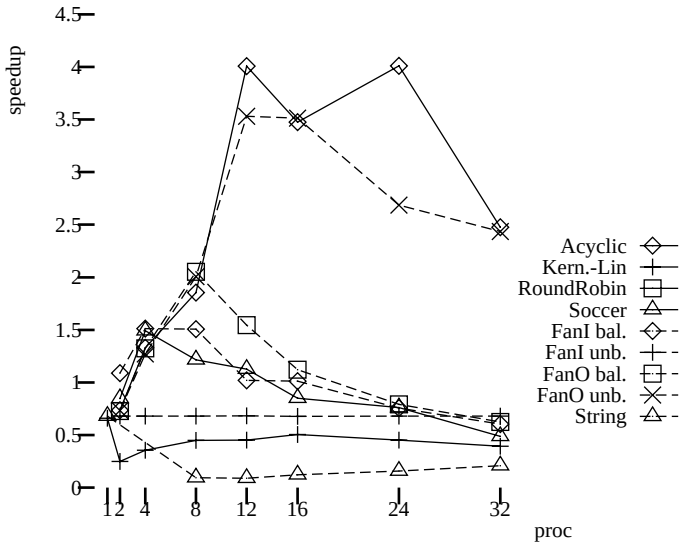


Abbildung 8.3: Speedup, s35932

Die Verfahren der unbalancierten Kegel und der azyklisch Aufteilung zeigen auf niedrigem Niveau stagnierende Ergebnisse. Der Einfluß der Kommunikation bei ersteren ist aufgrund der stark variierenden Partitionsgrößen gering. Die azyklische Methode versucht prinzipiell einigermaßen ausgewogene Partitionsgrößen zu erreichen, was aber bei Modell s13207 nicht gelingt. Es existiert wenigstens ein sehr großer Zyklus, der bei steigender Prozessoranzahl ein ansonsten mit dem konservativen Verfahren zu erwartendes gutes Verhalten einschränkt. Die Gesamtlaufzeit ist somit weitgehend durch die Laufzeit der diesen Teilgraphen enthaltenden Partition limitiert.

Modell s35932 weist prinzipiell ein ähnliches Verhalten auf wie die beiden ersten Schaltungen. Bis zu einer gewissen Prozessoranzahl steigen die Beschleunigungswerte an. Danach stellt sich eine Sättigung und die Stagnation der Speedups ein. Erfreulich ist hier jedoch, daß die parallelen Verfahren einen relevanten und merklichen Geschwindigkeitsgewinn erbringen. Das Wachstum des Speed-

up ist bis zu einer Prozessoranzahl von 12 Knoten zu beobachten und erreicht dabei Spitzenwerte zwischen 3,5 und 4 (unbalancierte Fan-out-Kegel und azyklisch). Selbst mit einigen balancierten Verfahren (Soccer und Kegel) können auf bis zu 8 Prozessoren Geschwindigkeitssteigerungen mit den Faktoren 1,5 und 2 verzeichnet werden. Danach sackt die Kurve wieder stark ab.

Anzahl Part.	Bewertungszahlen								
	RR	azykl.	Fan-in Cones		Fan-out Cones		K/L	Soccer	String
			unbal.	bal.	unbal.	bal.			
s13207									
1	15709	15709	15709	15709	15709	15709	15709	15709	15709
2	7855	7928	8366	7861	8033	7862	7855	7855	7855
4	3927	4107	7015	3932	6766	3975	3927	3927	3927
8	1964	2862	6325	1970	5917	2016	1964	1964	1964
12	1309	2505	6150	1319	5850	1354	1309	1309	1309
16	982	2343	6082	993	5820	1011	982	982	982
24	655	2166	6033	664	5782	668	655	655	655
32	491	2077	6006	497	5779	498	491	491	491
s35932									
1	40685	40685	40685	40685	40685	40685	40685	40685	40685
2	20343	20345	39985	20349	20432	20432	20343	20343	20343
4	10171	10420	39354	10177	10446	10459	10171	10171	10171
8	5086	5645	39354	5092	5460	5472	5086	5086	5086
12	3390	3930	39353	3469	4173	3807	3391	3390	3390
16	2543	3594	39353	2585	4016	2813	2543	2543	2543
24	1695	3315	39353	1718	3891	1842	1695	1695	1695
32	1271	3271	39353	1290	3887	1316	1271	1271	1271
64		3219						636	

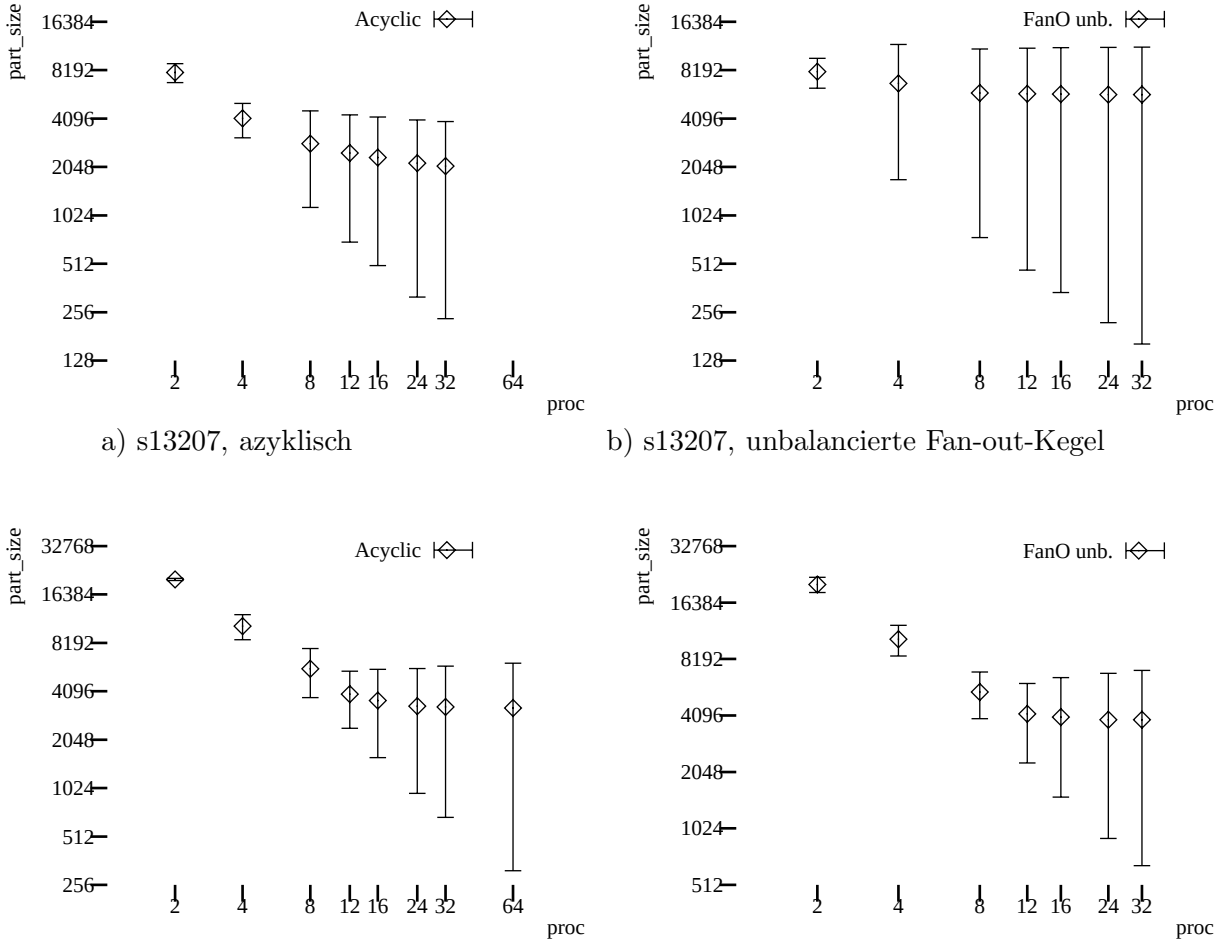
Tabelle 8.4: Klassifikation der Partitionierungsbalance

Mit zunehmender Modellgröße ergeben sich also wachsende Speedups, die bei gleichzeitiger Erhöhung der Partitionsanzahlen wieder abnehmen. Die Vermutung liegt somit nahe, daß in der betrachteten Applikationsklasse mit der verfügbaren Grundgranularität eine gewisse Mindestanzahl von Objekten auf einem Simulator (LP) vorhanden sein muß, um Beschleunigungen zu erreichen. Weiterhin scheint eine einigermaßen ausgewogene Verteilung der Objekte auf alle beteiligten parallelen Simulatorkomponenten nötig zu sein. Um diese Vermutungen genauer zu untersuchen, legt Tabelle 8.4 ein Maß fest, das den möglichen Einfluß der Partitionierung auf das Laufzeitverhalten beschreibt. Die für die einzelnen Verfahren p und Prozessoranzahlen n angegebenen Werte geben eine gewichtete Kenngröße für die Ausgewogenheit der Aufteilung wieder. Die Berechnung der Werte $w_{p,n}$ hängt von der Gesamtanzahl der Objekte n_{obj} im Simulationsmodell und der Anzahl der jeder Partition P_i zugewiesenen Objekte n_{P_i} ab und erfolgte nach folgender Formel:

$$w_{p,n} = \sum_{i=1}^{n} n_{P_i} * \frac{n_{P_i}}{n_{obj}}.$$

Größere Partitionen werden somit stärker gewichtet und haben einen maßgeblichen Einfluß auf den berechneten Endwert. Bei gleichgewichtigen Partitionen liegt $w_{p,n}$ nahe beim Idealwert $\frac{n_{obj}}{n}$.

Je mehr Partitionen in der Größe voneinander abweichen desto stärker nähert sich der Wert n_{obj} an. Große Werte stehen somit für die Existenz einer oder einiger weniger großer und vieler kleiner Partitionen.



a) s13207, azyklisch

b) s13207, unbalancierte Fan-out-Kegel

c) s35932, azyklisch

d) s35932, unbalancierte Fan-out-Kegel

Abbildung 8.4: Varianz der Partitionsgrößen (ausgewählte Beispiele)

Abbildung 8.4 zeigt darüber hinaus die maximalen Abweichungen der Partitionsgrößen vom berechneten Qualitätsmaß der Partitionen für die Partitionierungsverfahren azyklisch und unbalancierte Fan-out-Kegel an, die die besten Beschleunigungswerte besitzen. Insbesondere bei s13207 zeigt sich für die azyklische Partitionierung die Existenz eines großen Clusters, das die Berechnungsdauer der parallelen Variante dominiert. Obwohl die zyklenfreie Partitionierung als optimal für das konservative Protokoll mit Deadlockvermeidung betrachtet werden kann, bestimmt das Vorhandensein eines oder weniger sehr großer Zyklen die Laufzeit.

Ähnliche Beobachtungen beschreiben auch Kartik und Abraham [KAA92a] in ihrem zeitgesteuerten Parallelsimulator. Sie versuchen die Auswirkungen unterschiedlicher Partitionsgrößen durch Erhöhen der Pufferkapazitäten für Ereignisse in zukünftigen Zeitschritten zu mildern. Dadurch können Prozessoren bereits vorliegende Ereignisse der nachfolgenden Zeitschritte bearbeiten, auch wenn einzelne treibende Partitionen aufgrund ihrer Größe noch in einem vorhergehenden Zeitschritt aktiv sind. Für den Bereich der asynchronen parallelen Logiksimulation ist dieses Verfahren nicht anwendbar, da einerseits kein von Null verschiedener Lookahead garantiert werden kann und

andererseits die Ereignisse ohnehin bereits so früh wie möglich ausgeführt werden[8].

Der Einbruch der azyklischen Partitionierung bei Verwendung von mehr als 12 Partitionen läßt sich ebenfalls auf eine dominante Partition zurückführen. Insgesamt scheint die Schaltung s35932 eine sehr komplexe Struktur aufzuweisen, die jedoch weitgehend auf die Ansteuerung eines oder weniger Primärausgangssignale ausgelegt ist. Die unbalancierte Fan-in-Kegelpartitionierung generiert nur eine einzige relevante Partition während die restlichen Simulatoren nur einige wenige Objekte zugewiesen bekommen. Das erklärt auch die in Tabelle 8.3 auftretenden Laufzeiten, die mit der der Einprozessorpartitionierung bei String übereinstimmen und quasi eine verlangsamte sequentielle Simulation darstellen.

Abbildung 8.4 läßt sich entnehmen, daß eine Mindestgröße aller Partitionen nicht unterschritten werden darf. Vergleicht man die Beschleunigungswerte und die Partitionsgrößen, so fällt auf, daß die Speedup-Werte einbrechen, sobald die minimale Größe unter 3000 bis 5000 Gatter fällt. Bei der betrachteten Grundgranularität sind also mindestens Partitionen dieser Größenordnung notwendig, um moderate Beschleunigungen durch die Parallelisierung ereignisgesteuerter Simulation zu erreichen.

Bei den bisher noch nicht diskutierten Partitionierungsverfahren (Soccer, K/L, String) wurde versucht, die Anzahl der Elemente pro Simulator (unter der Annahme identischer Aktivitätsraten) zu balancieren. Bei diesem Ansatz sind die Prozessoren in etwa gleich stark mit Ereignisbearbeitung und Nachrichtenversand ausgelastet. Dennoch ergibt sich eine höhere Anzahl von Deadlocks und Ereignisnachrichten, so daß auch bei diesen Verfahren nur moderate Beschleunigungen und dieses auch erst ab einer gewissen Partitionsgröße erreicht werden.

Das Verhältnis zwischen der Anzahl simulierter Ereignisse pro verschickter Nachricht beziehungsweise der prozentuale Anteil der Gesamtereignisse, der zwischen zwei aufeinanderfolgenden Deadlocks verarbeitet wurde, ist in den Tabellen 8.5 und 8.6 aufgeführt[9].

Anzahl Part.	ausgeführte Events / Nachricht					
	s13207			s35932		
	Fan-out unb.	azykl.	Soccer	Fan-out unb.	azykl.	Soccer
2	1137542	1096306	4175	3319170	3215512	451
4	191319	151561	1957	996635	665632	655
8	106145	27696	1958	332583	167049	128
12	72391	13517	1324	111233	93782	104
16	54989	13158	180	83425	70335	89
24	36944	9751	266	55617	403	50
32	118126	7748	1373	41713	304	23
64					255	27

Tabelle 8.5: Anzahl der Events pro Ereignisnachricht

Je höher der Wert in Tabelle 8.5 ist, um so weniger Aufwand wurde für die Synchronisation der Teilsimulatoren untereinander verwendet. Bei großen Werten arbeiten die einzelnen parallelen Simulatoren nahezu autonom. Dies trifft für die Partitionierung mit Fan-out-Kegeln zu, da dort im

[8] Das trifft für den Bereich der geclusterten Partitionen zwar nicht vollständig zu, jedoch ist eine Pufferung von Ereignissen zukünftiger Zeitschritte auch nicht für beliebige Partitionierungsverfahren einsetzbar.

[9] Die dargestellten Werte wurden als Mittelwert über alle Partitionen berechnet. Sie ähneln den Bewertungsgrößen von Fujimoto [FUJ88b] und Reed, Malony und McCredie [RMM88a], die die Nullnachrichten- und Deadlockanzahl in Relation zur Gesamtanzahl aller Nachrichten setzten. Die hier verwendeten Verhältnisse drücken aber eher das Maß des Overheads in einem konservativen Verfahren aus als die beiden zitierten Bewertungskriterien.

wesentlichen die Berechnungen in einer sehr großen Partition lokal ohne notwendige Kooperation mit anderen Prozessoren erfolgen kann. Bei einer Gesamtereigniszahl von 3,8 bzw. 12 Millionen simulierten Ereignissen für s13207 und s35932 zeigt sich auch, daß die Soccer-Partitionierung keine sonderlich gute Realisierung darstellt. Insbesondere bei s35932 wird die Dauer des Versands einer Ereignisnachricht gerade durch die Ausführungszeit der lokalen Ereignisse mit Grundgranularität (Verhältnis 30:1) aufgewogen. Die Simulatoren sind wenigstens zur Hälfte mit Kommunikation beschäftigt. Bei der azyklischen Partitionierung bewegt sich der Aufwand für Kommunikation im Verhältnis zur Ereignisevaluierung zumindest bei geringeren Prozessoranzahlen noch innerhalb sinnvoller Grenzen. Dies spiegelt sich ja auch bei den erreichten Beschleunigungswerten wieder.

Das konservative Verfahren nach dem Deadlock-detection- und -recovery-Verfahren erfordert neben der Synchronisation durch Ereignisnachrichten aber auch noch weiteren Aufwand zur Behandlung von Deadlocks. Der dadurch verursachte Aufwand ist in Tabelle 8.6 beschrieben. Die Werte geben an, welchen Beitrag zur Bearbeitung aller Ereignisse jede einzelne Deadlockauflösung leistet. Informell ausgedrückt bedeutet das für jeden Eintrag k: Die Erkennung eines Deadlocks bewirkt die anschließende Ausführung von durchschnittlich k Prozent der Gesamtereignisse. Die unbalancierte Fan-out-Kegelpartitionierung und die azyklische Aufteilung der Schaltungen verhindert die Entstehung von Deadlocks, da alle Zyklen lokal auf einzelnen Prozessoren abgelegt werden. Es sind somit ohne zusätzlichen Aufwand zur Deadlockerkennung alle Ereignisse verklemmungsfrei ausführbar, sobald die Kanalzeiten die entsprechenden Werte erreichen.

Bei den Verfahren Soccer und Kernighan-Lin wird das Aufbrechen von Zyklen nicht berücksichtigt. Deshalb laufen die Simulatoren sehr oft in globale Verklemmungen, die explizit behandelt werden müssen. Bei den Untersuchungen wurde festgestellt, daß bei Verwendung dieser schnittkostenminimierenden Verfahren ein erheblicher Teil des Rechenaufwands für die Deadlockerkennung und -auflösung benötigt wird.

Anzahl Part.	Prozentualer Anteil aller Events / Deadlock							
	s13207				s35932			
	Fan-out unb.	azykl.	Soccer	Kern.	Fan-out unb.	azykl.	Soccer	Kern.
2	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
4	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
8	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
12	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
16	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
24	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
32	100.0000	100.0000	0.0077	0.0077	100.0000	100.0000	0.0121	0.0121
64						100.0000	0.0121	

Tabelle 8.6: Anteilsmäßige Ausführung der Gesamtereignisse pro Deadlock

Erstaunlicherweise werden bei den betrachteten realen Schaltungen unabhängig vom Partitionierungsverfahren und von der verwendeten Prozessoranzahl jeweils stets exakt dieselbe Anzahl von Deadlocks in jedem Simulationslauf beobachtet. Daher rühren auch die identischen Zahlenwerte in Tabelle 8.6. Es scheinen somit nur kleine Gruppen von Objekten mit zyklischer Koppelung für die Auslösung von Deadlocks verantwortlich zu sein. Werden Elemente einer solchen Gruppe auf mehrere Prozessoren verteilt, so wird sofort eine massive Deadlocklawine ausgelöst. Besitzt jeder der beteiligten Partner ein kleinstes auszuführendes Ereignis mit identischem Zeitstempel und kann jeder potentiell dem anderen weitere Ereignisse einplanen, so blockieren sie sich gegenseitig bei der Ausführung. Es kommt zum globalen Deadlock aufgrund eines einzigen Zyklus im gesamten Modell

(s. Abb. 2.13), sobald alle Ereignisse im System mit kleinerem Zeitstempel bearbeitet sind. Bei den realen Modellen kann diese Konstellation bereits durch Verteilung der beiden rückgekoppelten Gatter der Latchstufe eines Flipflops erzeugt werden[10].

Eine weitere Ursache von Deadlocks stellen selbst bei zyklenfreien Modellen die Existenz zweier unidirektionaler Nachrichtenkanäle zwischen zwei LPs dar, wenn diese LPs für die Simulation von Clustern aus mehreren Objekten zuständig sind. Zur Demonstration des Problems genügt eine Kette aus drei Objekten, wobei das erste dem zweiten Objekt Ereignisse einplant und dieses wiederum das dritte Objekt mit Ereignissen versorgt. Das erste und dritte Objekt wird auf LP_1 plaziert, das mittlere wird LP_2 zugewiesen. Da LP_1 auf mögliche Ereignisse von LP_2 für das dritte Objekt warten muß, darf es eventuell bereits vorhandene Ereignisse des ersten Objekts nicht ausführen. Dadurch erhält LP_2 keine neuen Garantien und beide Prozesse blockieren. Man erhält somit aufgrund der Clusterung ein identisches Verhalten wie bei der zyklischen Variante. Bei einigen längeren Simulationsläufen mit zyklenfreien synthetischen Modellen (inv10000_chain, inv64_8_branch) wurden insbesondere auf diese Art ausgelöste Deadlocks beobachtet und für verschiedene Partitionierungen oder Prozessoranzahlen dann auch variierende Deadlockanzahlen festgestellt.

Die verwendeten Schaltkreise können im Kontext der kommunikationsminimierenden Partitionierungen nur als Beispiele verstanden werden. Wird die Reihenfolge der Objekte in der Modellbeschreibung modifiziert, so kann ggf. eine vollständig andere Aufteilung der Schaltung generiert werden. Die negativen Charakteristika, die bei den drei betrachteten Modellen uniform beobachtet wurden, lassen jedoch vermuten, daß diese Verfahren für eine effiziente Simulation nicht sonderlich geeignet sind. Die anderen Algorithmen berücksichtigen die Topologie des Modellgraphen und generieren somit stets Aufteilungen mit wenigstens ähnlichem Aufbau, auch wenn die Partitionen nicht unbedingt identisch sind.

8.1.1.2 Synthetische Modelle

Bei den synthetischen Benchmarks können die Partitionierungsverfahren entsprechend den nicht größenbalancierten Fan-in- und Fan-out-Kegeln ignoriert werden. Sie generieren stets unabhängig von der Knotenanzahl lediglich eine einzige Partition[11].

Anzahl Part.	Simulationsdauer			
	Fan-in Cones bal.	Fan-out Cones bal.	K/L	Soccer
2		16,61	16,69	16,69
4		9,80	9,78	10,26
8		5,95	5,99	703,54
16	4,56	4,06	1462,56	1519,32
24	5,02	4,64	2595,45	2635,11
32	4,26	4,13	3376,80	3744,44

Tabelle 8.7: Simulationsdauer mit konservativem Verfahren, inv64_chain

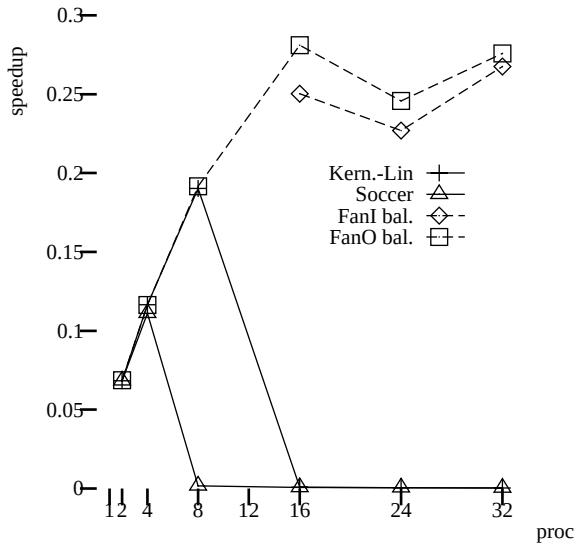
Die Aufteilungen bei der azyklischen und bei der String-Methode entsprechen im wesentlichen der mit balancierten Cone-Methoden generierten. Im Falle von rückgekoppelten Schaltungen

[10] Tatsächlich wurde dieses Aufsplitten z.B. bei der Stringpartitionierung für die Modelle s13207 und s35932 beobachtet.

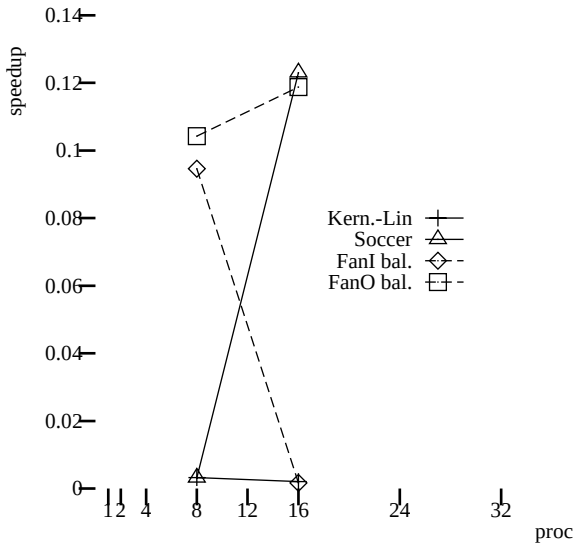
[11] Mit Ausnahme der `Branch`modelle.

`invn_cycle` und `invn_k_cs` macht die zyklenfreie Aufteilung keinen Sinn, da alle Objekte in einer einzigen Schleife enthalten sind. Deshalb wurden im folgenden nur die beiden balancierten Kegelverfahren sowie K/L und Soccer betrachtet.

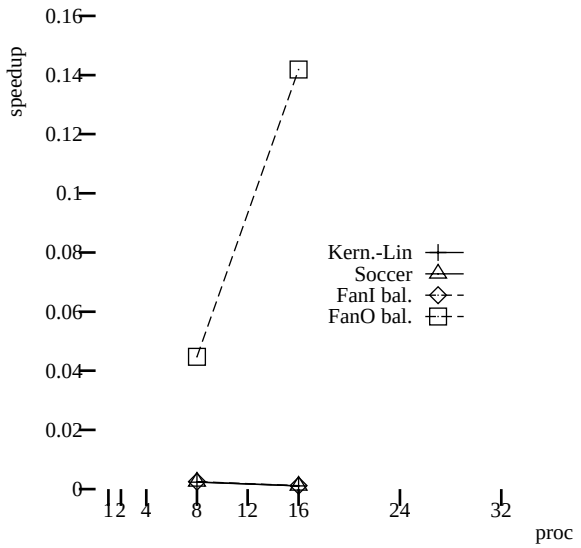
Die Laufzeiten der Simulationen sind für zwei verschieden große Schaltungen in den Tabellen 8.7 und 8.8 dargestellt. Die Ergebnisse ähneln auch hier stark denen bei den realen Schaltungen. Die Partitionierungen, die die Topologie der Schaltung berücksichtigen, generieren beim Start mit zunehmender Prozessoranzahl für die kleinen Schaltungen relativ zur Simulation auf weniger Prozessoren scheinbaren Speedup. Die absoluten Werte in Relation zum sequentiellen Simulationslauf zeigen allerdings vernichtende Geschwindigkeitseinbußen. Bei den kommunikationsminimierenden Aufteilungen (K/L, Soccer) steigt die Laufzeit der parallelen Simulation meist bereist von Anbeginn mit zunehmender Knotenzahl.



a) Chain



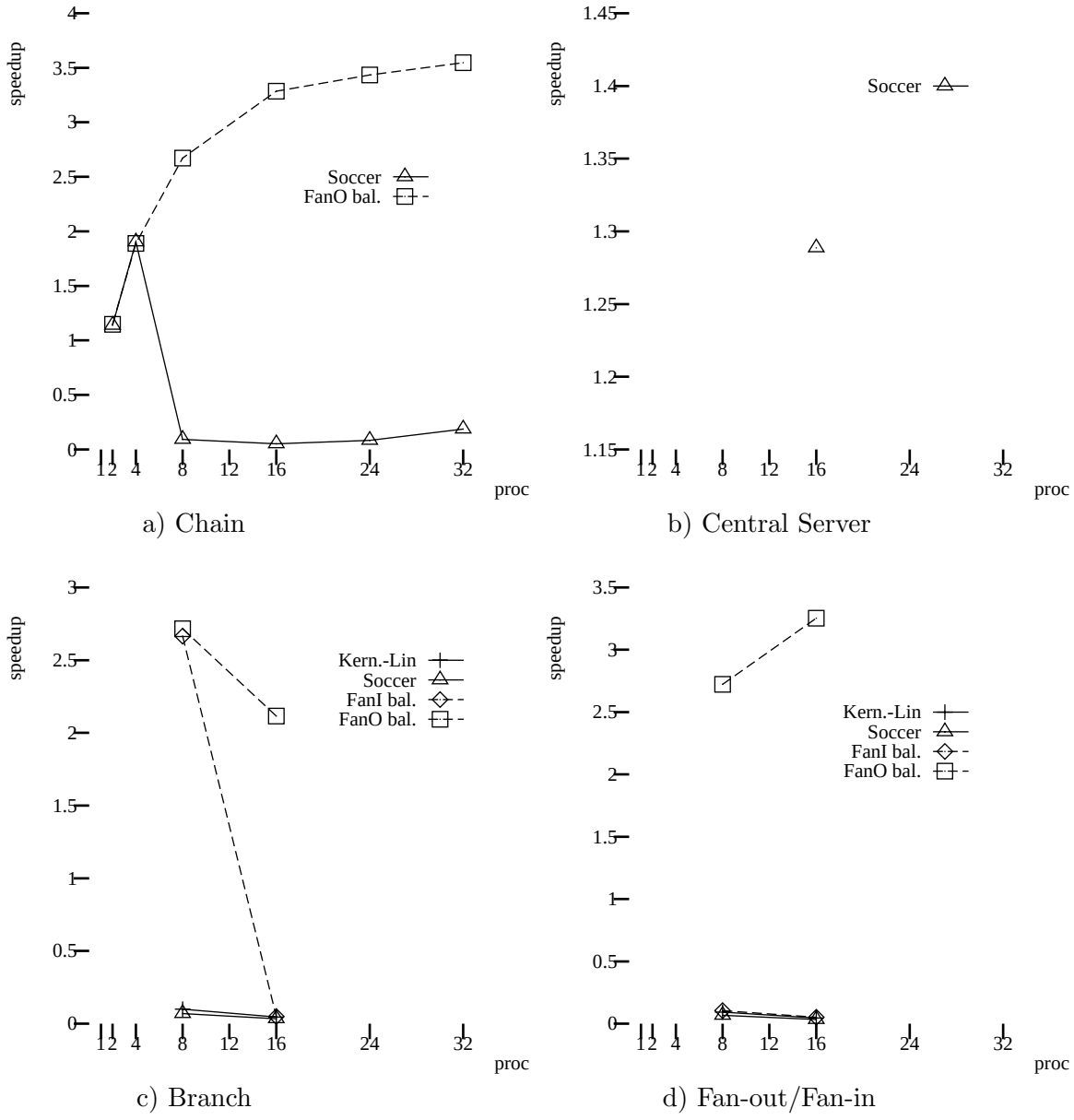
b) Branch



c) Fan-out/Fan-in

Abbildung 8.5: Speedup, inv64*

Die größeren Benchmarks erreichen wesentlich bessere Werte, die aber dennoch unterhalb der erhofften linearen Beschleunigung liegen. Die Fan-out-Kegelpartitionierung generiert z.B. für Modell inv10000_chain trotz blockweiser Anordnung der Objekte als Pipelinestufen lediglich zwischen 10 und 20 Prozent der theoretisch zu erwartenden Speedup-Werte (Abb. 8.6). Für die zehnmal größere Schaltung inv100000_chain (Abb. 8.7) wird mit demselben Partitionierungsverfahren immerhin eine Effizienz von 30 Prozent erreicht.



a) Chain b) Central Server

c) Branch d) Fan-out/Fan-in

Abbildung 8.6: Speedup, inv10000*

Bei allen anderen künstlichen Modellen ergaben sich ähnliche Tendenzen, weshalb auf die Wiedergabe der reinen Laufzeiten in Tabellenform verzichtet werden kann und lediglich die Trends in den Graphen (Abb. 8.5 und 8.6) für den erreichten Speedup dargestellt werden.

Anzahl Part.	Simulationsdauer	
	Fan-out bal.	Soccer
1		1177,61
2	871,57	871,71
4	512,48	507,81
8	271,91	1510,06
16	143,81	
24	100,61	
32	82,60	

Tabelle 8.8: Simulationsdauer mit konservativem Verfahren, inv100000_chain

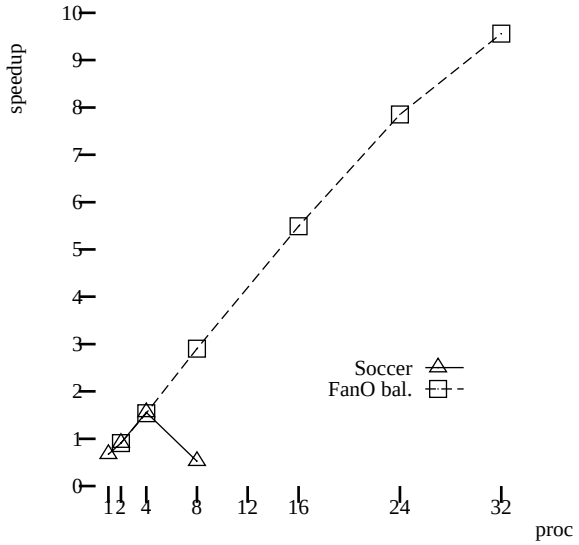


Abbildung 8.7: Speedup, inv100000_chain

8.1.2 Übereinstimmung mit der Leistungsvorhersage der CPA

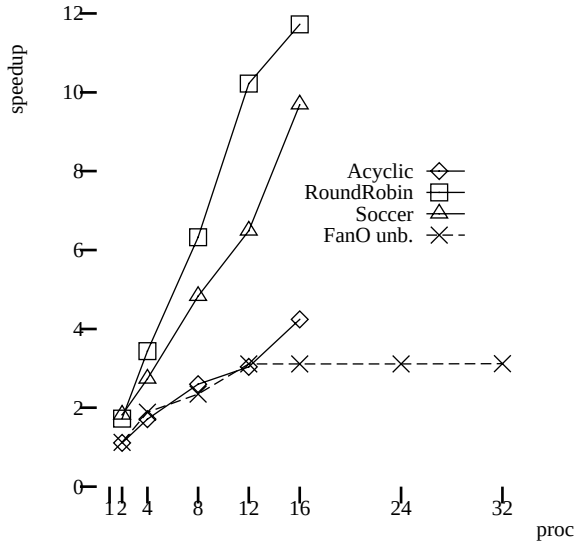
In Kapitel 7.4.1 wurden anhand der Kritische-Pfad-Analyse Versuche über die maximal mögliche Beschleunigung der betrachteten Schaltungen unter der Annahme unbegrenzter Ressourcen durchgeführt. Die dort gefundenen Ergebnisse weichen gravierend von den unter realen Bedingungen mit konservativen Methoden erreichten Resultaten ab. In diesem Abschnitt soll es nun darum gehen, das theoretisch vorhandene Parallelisierungspotential unter Einbeziehung verschiedener Partitionierungen zu untersuchen, um somit ein realistischeres Maß der Beschleunigbarkeit zu erhalten. Dabei wird zunächst die Berechnung des kritischen Pfads unter Vernachlässigung der Kommunikationszeit durchgeführt. Anschließend werden auch verschieden hohe Aufwände für die Kommunikation in dieses Kostenmodell integriert.

8.1.2.1 Reine CPA ohne Kommunikation

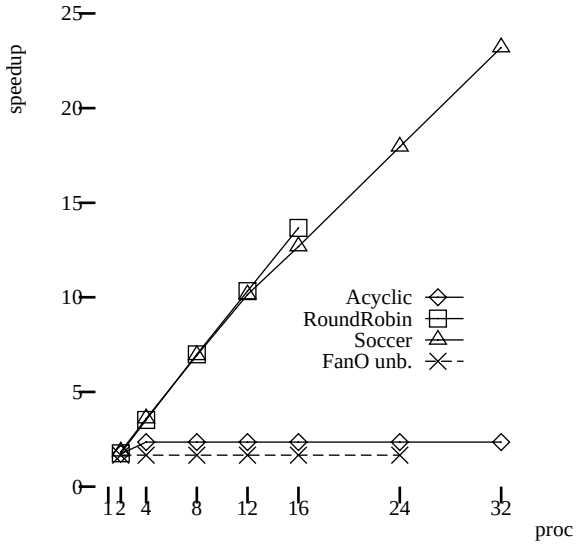
Partitionierungsverfahren, bei denen eine Balancierung in Hinblick auf die Objektanzahl vorgenommen wird, schneiden bei den CPA-Berechnungen extrem gut ab. Relativ zur Prozessoranzahl erreichen die Bewertungen Effizienzvoraussagen von 70 bis 75 Prozent.

Die bei den realen Messungen aufgetretenen Verlangsamungen insbesondere bei kommunika-

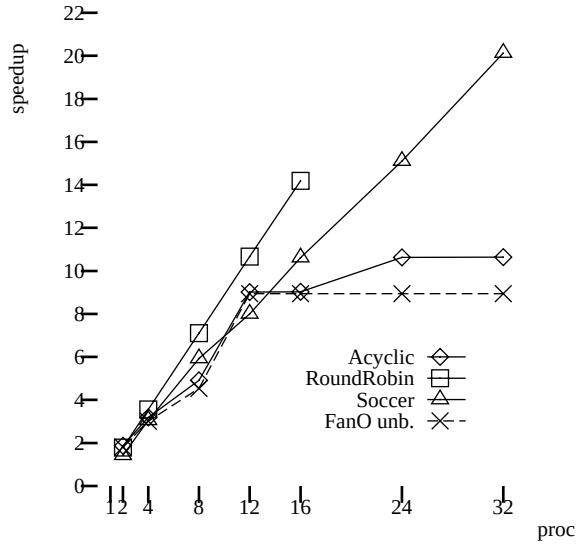
tionsaufwendigen Partitionierungen wie Round-robin konnten hier nicht beobachtet werden. Aufgrund der Clusterung einer großen Anzahl von Objekten auf den Prozessoren sind diese theoretisch die überwiegende Zeit ausgelastet. Die relevanten Kommunikationskosten, die das konservative Verfahren drastisch bremsen, werden bei der CPA-Methode zunächst ignoriert. Diese Ergebnisse zeigen, daß eine alleinige Betrachtung des kritischen Pfads ohne Berücksichtigung der Kommunikationskosten auf einer Parallelrechnerumgebung mit verteiltem Speicher nicht ausreicht, um eine adäquate Voraussage des vorhandenen Parallelisierungsvermögens machen zu können.



a) s1196



b) s13207



c) s35932

Abbildung 8.8: Kommunikationsfreie CPA mit Partitionierung

Allerdings fällt auch bei der Kritische-Pfad-Analyse das extreme Verhalten von Schaltungen auf, die extrem große Zyklen haben. Der mit der azyklischen Partitionierung selbst mit der CPA

erreichbare Speedup liegt für Modell s13207 bei wachsender Prozessoranzahl stets und nahezu konstant unterhalb von 3 (Abb. 8.8b), da die Simulationszeit wie bereits beim parallelen Ablauf von der sequentiellen Abarbeitung innerhalb einer entsprechend großen Partition dominiert wird. Dies gilt in verstärktem Maße auch für die unbalancierte Fan-out-Kegelpartitionierung, die einen recht großen Kegel ab einer gewissen Partitionisanzahl nicht weiter zerlegt und somit zu einer die Simulationsdauer bestimmenden Partition führt.

Die Existenz weniger langer Zyklen, die die bei balancierter Aufteilung zu erwartende Partitionsgröße wesentlich überschreiten, läßt also bereits auf ein niedriges Parallelisierungspotential einer Schaltung bei Verwendung konservativer Verfahren schließen, da dann entweder:

- eine dominante Partition existiert, die die Laufzeit derjenigen der sequentiellen Simulation annähert, oder
- viele Deadlocks (über Prozessorgrenzen hinweg) auftreten können, da Zyklen nicht lokal behandelt werden, oder
- viele Nullnachrichten zur Vermeidung der Deadlocks innerhalb prozessorübergreifender Zyklen benötigt werden.

Mit Hilfe der zyklenfreien Partitionierung läßt sich das Vorhandensein einer solchen Partition leicht nachweisen. Ergibt die CPA-Berechnung bereits ohne Berücksichtigung der Kommunikation sehr schlechte CPA-Werte, so ist in einem realen konservativen Verfahren auch nicht mit hohen Beschleunigungen zu rechnen.

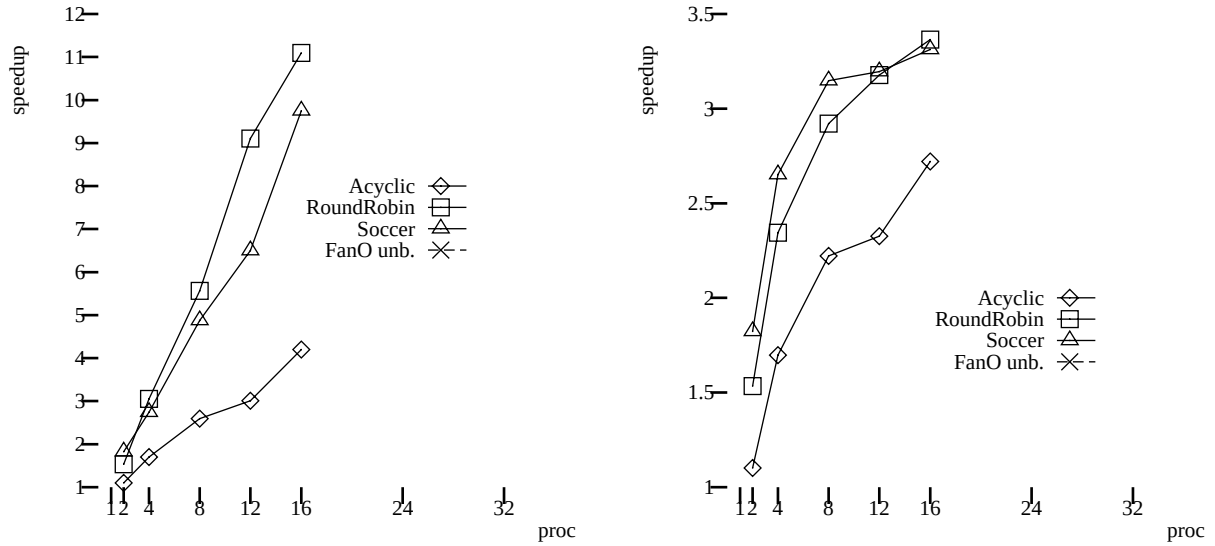
Die von der reinen CPA prognostizierte Geschwindigkeitssteigerung ist nicht unbedingt gerechtfertigt. Sie geht von in der Praxis nicht realisierbaren Annahmen bezüglich der Kommunikationshardware aus. Lediglich als Ausscheidungskriterium wie im Falle der azyklischen Partitionierung für Schaltung s13207 kann die CPA bereits ohne Bewertung der Kommunikation maßgeblich sein.

8.1.2.2 Berücksichtigung konstanter Kommunikationsdauer für CPA

Die Berücksichtigung realer Kommunikationskosten bei der Abschätzung der Speedup-Möglichkeiten scheint also unabdingbar für eine gerechtfertigte Beurteilung. Die Gewichtungsfaktoren zwischen Kommunikations- und Ereignisgranularität werden wie im Fall einer virtuell unbeschränkten Prozessoranzahl so gewählt, daß einmal Kommunikation und Berechnung im Gleichgewicht stehen und andererseits die Kommunikation die Überhand erhält. Die bisher betrachteten Ergebnisse stehen für ein extremes Überwiegen der Ereignisberechnungen.

In den Betrachtungen wurde aufgrund der in Tabelle 7.9 beschriebenen Verzerrungen bei Messungen mit der Grundgranularität von 10 μs pro Ereignis eine künstliche Erhöhung der Granularität auf 300 μs vorgenommen. Bei Gleichgewicht zwischen Berechnung und Kommunikation wird die Nachrichtenlaufzeit somit mit demselben Wert belegt. Beim Überwiegen des Kommunikationsaufwands wird die Kommunikationsdauer (entsprechend dem realen Fall auf dem GC/PP) mit dem 30-fachen Wert (hier also 9 ms) veranschlagt.

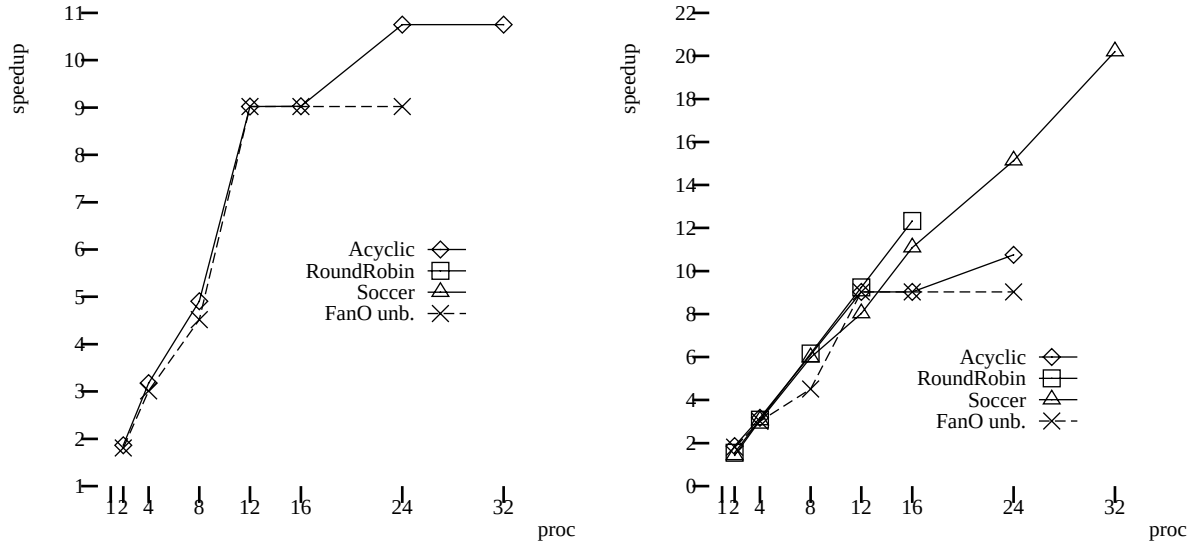
Abb. 8.9a zeigt, daß bei Gleichgewichtung der beiden Parameter die Beschleunigungsvorhersagen für das kleine Modell s1196 immer noch unrealistisch hohe Werte liefern. Erst bei Verwendung des realen Verhältnisses brechen die Beschleunigungsvorhersagen ein (Abb. 8.9b). Sie liegen zwar immer noch stark oberhalb der im parallelen Verfahren beobachteten Laufzeiten, geben aber schon eine etwas desillusioniertere Perspektive auf die Beschleunigungsmöglichkeiten der parallelen Simulation wieder.



a) Gleichgewicht b) Reale Kommunikationskosten

Abbildung 8.9: Kommunikationsbehaftete CPA mit Partitionierung, s1196

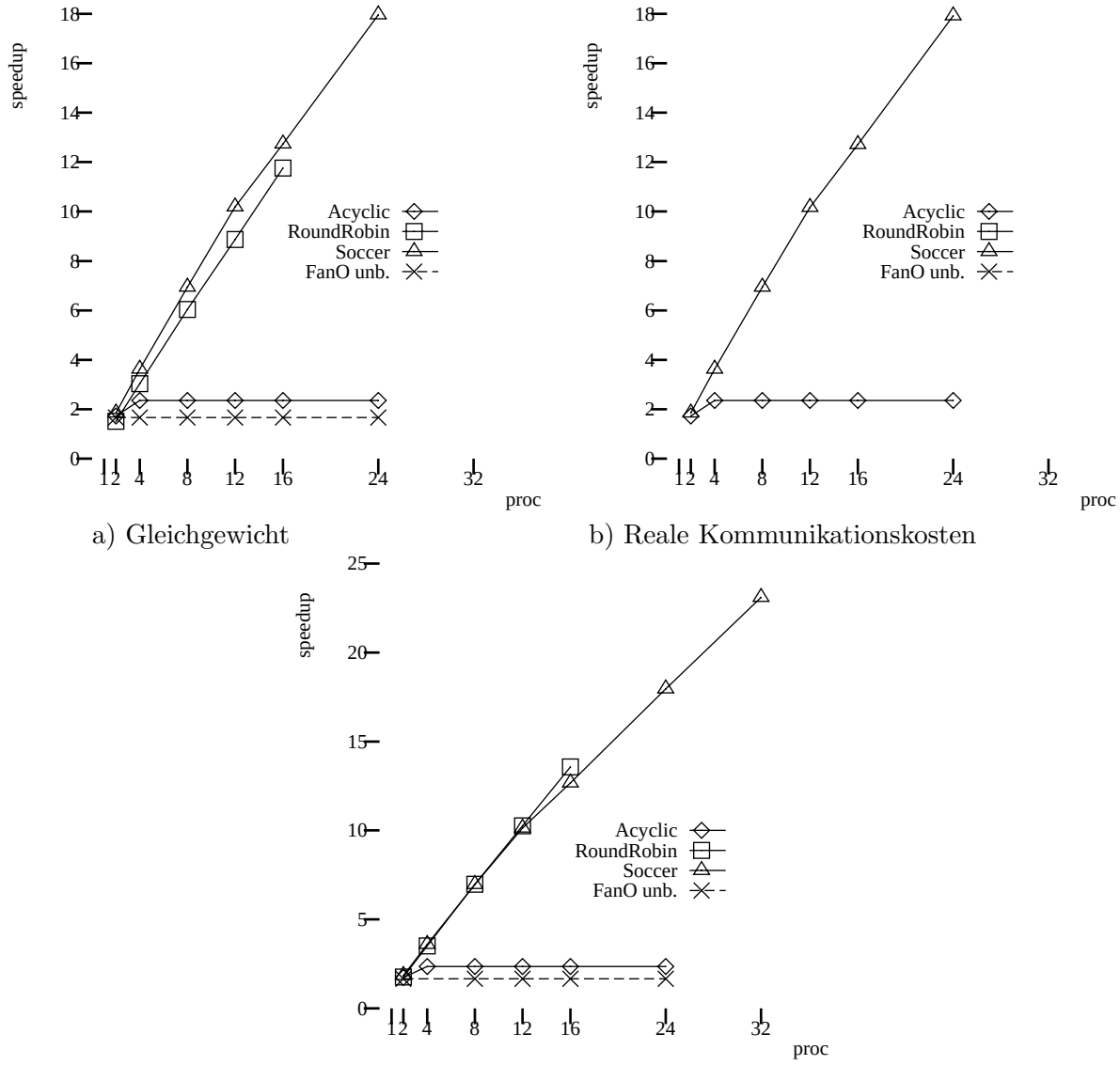
Die beiden größeren Modelle s35932 und s13207 zeigen dagegen auch bei Berücksichtigung der Kommunikationsdauer ein unverändertes Verhalten (Abb. 8.10 und 8.11). Um auszuschließen, daß diese Ergebnisse durch die Erhöhung der Granularität ungerechtfertigt hoch ausfallen, wurde für s13207 eine Kontrollmessung mit unveränderter Ereignisbearbeitungsdauer und Kommunikationsdauer 300 μs durchgeführt. Die beobachteten Ergebnisse in Abb. 8.11b und c sind jedoch identisch, so daß diese Parameterwahl als Einflußfaktor ausscheidet. Das CPA-Verfahren sagt selbst bei Einbeziehen der Kommunikation eine erhebliche Beschleunigung voraus, die mit den realen parallelen Verfahren nicht erreicht werden konnte.



a) Gleichgewicht b) Reale Kommunikationskosten

Abbildung 8.10: Kommunikationsbehaftete CPA mit Partitionierung, s35932

Das recht seltsame Voraussageverhalten der CPA wurde am Beispiel der Soccer-Partitionierung für das Modell s13207 weiter untersucht. Insgesamt erreicht das Soccer-Verfahren das ursprünglich in es gesetzte Ziel, die Anzahl der Nachrichten zu minimieren, recht gut. Bei den Untersuchungen des konservativen Verfahrens liegt das Verhältnis von Ereignissen zu Nachrichten bei wenigstens 30:1 für die Soccerpartitionierung[12].



a) Gleichgewicht

b) Reale Kommunikationskosten

c) Reale Kommunikationskosten (bei Granularität 10 μs)

Abbildung 8.11: Kommunikationsbehaftete CPA mit Partitionierung, s13207

Die mittels CPA prognostizierten Beschleunigungswerte brechen bei der Berücksichtigung der Kommunikation erst ein, wenn der Aufwand zwischen Kommunikation und Berechnungskosten das Verhältnis 400:1 erreicht, d.h. der Versand einer Nachricht sich in der Größenordnung von 4 ms bewegt. Ab einer Nachrichtengranularität von 25 ms kommt der Speedup gänzlich zum Erliegen.

[12]Dieser Wert stammt aus der Messung mit 16 Prozessoren; bei Verwendung von 8 Prozessoren ergibt sich ein besseres Verhältnis von ca. 250:1.

Die Ergebnisse decken sich bei Messungen unter Verwendung der Ereignislängen von 10 μs und 300 μs.

Modell s35932 weist ein im Sinne der Parallelisierbarkeit ähnliches Verhältnis zwischen Nachrichten und Ereignisauswertungen auf. Der Break-even-Wert zwischen sequentieller und vorhergesagter Leistung der parallelen Verfahren liegt hier später als für Schaltung s13207 bei einer Nachrichtengranularität von 100 ms. Der Einbruch der übersteigerten CPA-Werte findet ebenfalls erst ab einer simulierten Nachrichtenlaufzeit von 15 ms statt.

8.1.2.3 Zusatzaufwand durch konservative Verfahren

Da sich die realen Kommunikationskosten tatsächlich aber nicht in diesen Größenordnungen von wenigen Mikrosekunden bewegen sondern mehrere Hundertstel Mikrosekunden bis einige Millisekunden betragen, stellt sich nun die Frage, warum die Voraussagen für die zu erwartenden Beschleunigung bei Modellierung des realen Kommunikationsverhaltens dermaßen unzuverlässige Ergebnisse liefert. Zur Klärung wurden die maximale und mittlere Blockadedauer aufgrund fehlender Garantien für die einzelnen LPs und die Zeit zur Deadlockauflösung betrachtet (Tabelle 8.9).

Modell	Partitionierung		Blockadeanteil an der Gesamtdauer		Deadlockanteil an der Gesamtdauer
	Alg.	Knoten	max.	avg.	
s13207	a	8	35	6	0
s35932	a	8	50	14	0
s13207	a	16	37	3	0
s35932	a	16	55	19	0
s13207	c	8	3	1	0
s35932	c	8	97	12	0
s13207	c	16	92	6	0
s35932	c	16	97	12	0
s13207	D	8	69	42	18
s35932	D	8	29	20	0
s13207	D	16	77	39	21
s35932	D	16	60	32	15
s13207	k	8	33	21	9
s35932	k	8	12	8	3
s13207	k	16	52	25	19
s35932	k	16	26	14	6
s13207	s	8	56	32	15
s35932	s	8	32	20	8
s13207	s	16	63	33	22
s35932	s	16	46	25	11

Tabelle 8.9: Blockade- und Deadlockauflösungsanteil, reale Schaltungen

Gemessen wurde bei der Deadlockauflösung die Zeit, die benötigt wird, um über den Versand je einer Nachricht vom Kontrollprozeß an jeden beteiligten LP die Verklemmung aufzulösen. Der Aufwand für die eigentliche Erkennung läßt sich nicht klar erfassen, da eine Verklemmung schrittweise alle Prozessoren einbezieht und erst bei der Ankunft des Tokens beim Kontrollprozeß nach wenigstens einer Runde eine Aussage über den Zustand des Systems getroffen werden kann. Da

bei der Freigabe jedoch auch schrittweise die informierten Prozessoren wieder in einen aktiven Zustand wechseln noch bevor alle LPs von der Auflösung Kenntnis erhalten, repräsentiert die Zeit zur Deadlockauflösung auch teilweise die Kosten der Verklemmungsphase. Die Werte stellen somit untere Grenzen für den jeweiligen Anteil an der Gesamtlaufzeit dar.

Man erkennt an den maximalen Blockadeanteilen, daß einzelne Simulatoren zum Teil über 90 Prozent der Zeit blockieren und selbst bei niedrigeren Blockadeanteilen weitere 10 bis 20 Prozent der Laufzeit alleine zum Auflösen der Deadlocks benötigen. Insbesondere ist die Zeit vor Eintreten und Erkennen der globalen Deadlocks, in der die Parallelität mitunter drastisch sinkt, in letzterer Größe nicht enthalten. Sie wird nur zum Teil von der Blockadezeit abgedeckt.

Bei azyklischen Partitionierungen (a), die keine Deadlocks zulassen, blockieren einzelne Simulatoren über 50 Prozent der Laufzeit. Erstaunlich sind die hohen Blockadezeiten bei nichtbalancierten Partitionierungen. Die teilweise extremen Maximalwerte bei Kegelpartitionierung (c und D) sind jedoch nicht unbedingt auf die mangelnde Anzahl zugewiesener Objekte zurückzuführen. Die auf den betroffenen Simulatoren ausgeführten Ereignisse entsprechen bei s13207 weitgehend dem Durchschnitt aller Prozessoren oder liegen sogar darüber. Lediglich bei den Fan-in-Kegel-Verfahren (c) von s35932 konzentrieren sich die Objekte auf eine extrem große Partition.

Auch bei der Soccer-Partitionierung (s), die allen LPs die gleiche Anzahl von Objekten zuordnet, finden sich relative hohe Wartezeiten bei einzelnen Simulatoren bei ähnlich ausgewogener Ereignisverteilung.

Es fällt bei diesen vielfältigen Aspekten und Beobachtungen nicht leicht, eine allgemeine Schlußfolgerung zu ziehen. Der Aufwand durch die Blockade an Kommunikationskanälen mit nicht ausreichenden Garantien spielt auf jeden Fall eine nicht zu vernachlässigende Rolle bei dem betrachteten konservativen Simulationsverfahren. Ebenso ist der zu erbringende Overhead für die Deadlockauflösung ein gewisser Kostenfaktor, der durch eine ungünstige Partitionierung (z.B. Zulassen von prozessorübergreifenden Zyklen) stark beeinflußt werden kann.

Die Untersuchungen zeigen insgesamt, daß die Aufteilung der Schaltungen ein wichtiger Einflußfaktor auf die Leistungsfähigkeit der parallelen Simulation darstellt. Sie ist jedoch nicht die einzige maßgebliche Größe, wie die Betrachtungen bezüglich der Nachrichtenlaufzeiten bei der CPA-Analyse erkennen lassen.

8.1.3 Erhöhung der Granularität

Angesichts der bescheidenen Beschleunigungsmöglichkeiten des konservativen Basisverfahrens trotz sehr guter CPA-Prognosen stellt sich die Frage, ob die niedrige Granularität der Applikationsklasse Ursache dieses Verhaltens ist. Eventuell lassen sich für Simulationsläufe komplexerer Anwendungen, die eine wesentlich aufwendigere Funktion zur Berechnung der Ereignisse besitzen, bessere Ergebnisse erzielen. Aus diesem Grund wurden die Messungen aus Abschnitt 8.1.1 mit künstlich erhöhten Granularitäten bei der Ereignisbearbeitung analog zu Kapitel 7.3.3 durchgeführt. Die Untersuchungen erfolgten wiederum mit ausgewogener Bearbeitungsdauer zwischen Ereignissen und Nachrichten und mit einem starken Überhang der Eventgranularität (5 : 1). Die Ergebnisse werden für die realen Benchmarks s1196, s13207 und s35932 in den Tabellen 8.10 – 8.12 dargestellt.

Bei sequentiellen Laufzeiten von 61 bzw. 316 Sekunden erreichten die Experimente mit dem kleinen Modell s1196 bereits bei einer gleichgewichtigen Verteilung der Granularitäten Beschleunigungen bis zum Faktor 4 bei 8 verwendeten Prozessoren. Solche Ergebnisse wurden im Basisverfahren erst mit der größten untersuchten Schaltung s35932 erreicht. Eine weitere Erhöhung der Granularität lieferte keine weiteren Verbesserungen. Typischerweise wird aber hier wie auch bei den nachfolgend betrachteten Benchmarks wieder ein Leistungseinbruch bei steigender Prozessoranzahl

beobachtet, der sich im Rückgang der Beschleunigungswerte zeigt. Diese Stagnation tritt jedoch erst ab einer Knotenzahl von 8 auf und erfolgt auf einem wesentlich höheren Niveau.

Anzahl Part.	Simulationsdauer							
	RR	azykl.	Fan-in-cones		Fan-out-cones		K/L	Soccer
			unbal.	bal.	unbal.	bal.		
Ausgeglichene Granularität								
2	93,68	56,74					59,02	50,92
4	101,02	36,22	27,42	30,38	32,67	81,39	81,69	60,02
8	141,84	23,76	15,67	111,40	26,12	119,71	129,22	113,07
16	252,81	28,02	23,56	207,71	32,54	230,34	220,57	234,74
32		30,80	15,90	472,09	39,91	629,91		538,30
Überwiegen der Ereignislängen								
2	265,15	288,49					201,09	192,72
4	181,91	187,29	144,65	160,29	170,12	223,92	177,17	156,10
8	197,12	124,36	81,40	226,25	137,62	233,02	199,48	167,19
16	292,45	142,53	118,52	303,05	166,77	316,90	262,92	271,78

Tabelle 8.10: Simulationsdauer mit variierter Granularität, s1196

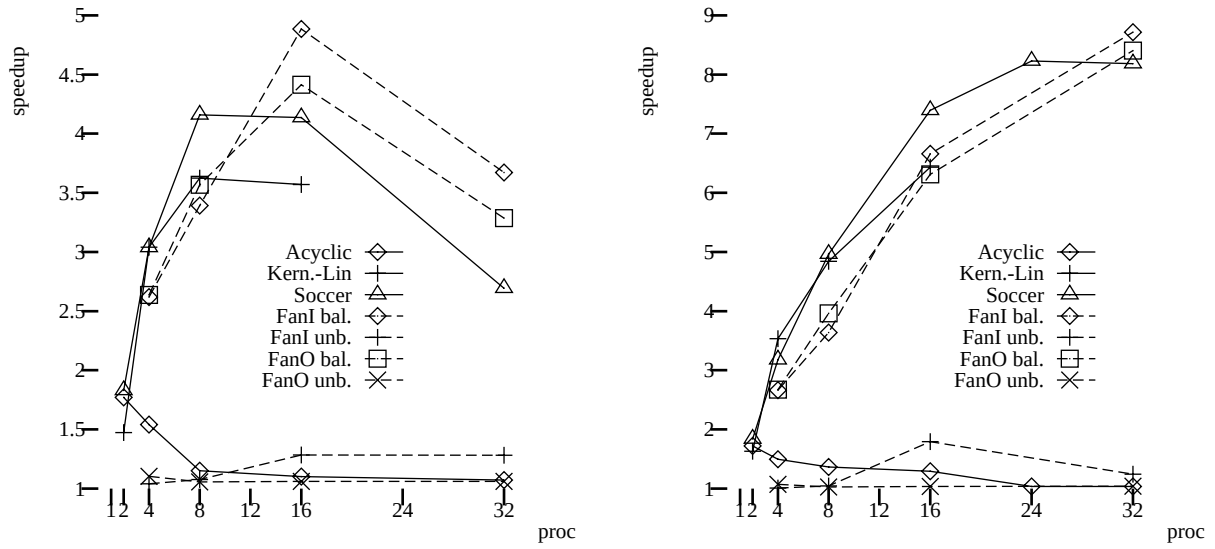
Die Speedup-Graphen (Abb. 8.12) des mittleren Modells s13207 folgen in ihrem tendenziellen Verhalten zum einen ebenfalls dem Simulationsverhalten mit Grundgranularität und andererseits den bei s1196 mit verlängerter Ereignisbearbeitungszeit beobachteten Trends. Die Beschleunigungsfaktoren steigen hier bis zu Werten von 5 auf 16 Prozessoren um anschließend wieder abzusinken. Bei weiterer Erhöhung der Ereignisbearbeitungsdauer ist jedoch bei den größenbalancierten Partitionierungen ein weiteres Ansteigen auf Werte zwischen 8 und 9 mit 32 Prozessoren zu beobachten.

Anzahl Part.	Simulationsdauer						
	azykl.	Fan-in-cones		Fan-out-cones		K/L	Soccer
		unbal.	bal.	unbal.	bal.		
Ausgeglichene Granularität							
2	964,15					1158,38	932,95
4	1107,04	1639,52	651,78	1546,24	647,51	561,34	561,75
8	1484,65	1586,19	503,37	1615,88	478,58	470,97	410,63
16	1550,73	1329,46	349,51	1608,66	386,80	478,28	412,81
32	1593,42	1331,86	465,10	1609,27	519,42		633,65
Überwiegen der Ereignislängen							
2	5070,50					5349,46	4748,28
4	5835,45	8648,55	3278,15	8150,05	3267,11	2471,22	2749,12
8	6394,64	8359,89	2401,34	8511,33	2204,37	1803,07	1759,89
16	6748,04	4864,68	1311,29	8414,40	1383,73	1357,58	1181,00
24	8438,67						1061,31
32	8425,79	7020,34	1001,96	8407,16	1039,09		1067,34

Tabelle 8.11: Simulationsdauer mit variierter Granularität, s13207

Durch die Aufblähung des Berechnungsaufwands fallen die Kommunikationszeiten und die Blockadezeiten sogut wir nicht mehr ins Gewicht. Die Nachrichten können entsprechend schnell

zugestellt werden, weil bei einer Balancierung der Objekte weitgehend alle Prozessoren stets beschäftigt sind und die Erhöhung der Garantien scheinbar schnell genug voranschreitet. Lediglich bei der Soccer-Partitionierung ist ein leichtes Abflachen der Kurve ab 16 Prozessoren zu bemerken.



a) Gleichgewicht b) hoher Berechnungsaufwand

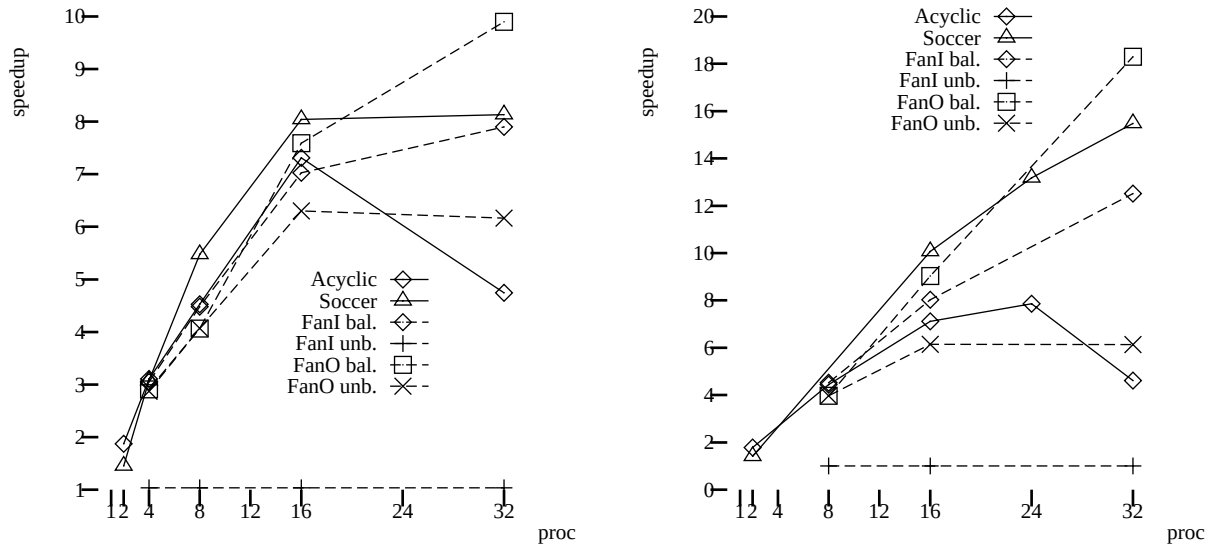
Abbildung 8.12: Gegenüberstellung verschiedener Granularitäten für Modell s13207

Die unbalancierten Verfahren schneiden dagegen schlecht ab. Mit der azyklischen Variante ist ohnehin kein Speedup oberhalb von 2,5 zu erwarten (siehe CPA-Ergebnisse). Die Kegelverfahren versagen ebenfalls bei Erhöhung der Granularität. Allerdings erreichen alle Methoden mit einer niedrigeren Simulationsdauer als die Laufzeiten des sequentiellen Simulators mit 1707 bzw. 8734 Sekunden auch bei größeren Knotenanzahlen Speedup-Werte, die zwar nur marginal aber dennoch oberhalb von "1" liegen.

Anzahl Part.	Simulationsdauer					
	azykl.	Fan-in-cones		Fan-out-cones		Soccer
		unbal.	bal.	unbal.	bal.	
Ausgeglichene Granularität						
2	2881,63					3720,57
4	1737,16	5199,97	1768,52	1877,64	1854,26	1752,35
8	1191,13	5197,37	1203,95	1324,93	1327,57	985,15
16	737,40	5203,38	767,36	855,59	710,32	670,70
32	1137,11	5200,66	682,75	874,88	544,79	663,09
Überwiegen der Ereignislängen						
2	15292,09					19362,73
8	6142,86	27314,23	6026,97	6869,02	6876,41	
16	3833,32	27335,13	3396,24	4442,98	3021,59	2710,05
24	3470,89					2070,66
32	5911,02	27322,79	2180,14	4449,18	1490,77	1762,01

Tabelle 8.12: Simulationsdauer mit variierter Granularität, s35932

Die Resultate der Experimente mit dem größten Modell s35932 schließen sich den bisherigen Beobachtungen an. Durch die Erhöhung der Ereignisgranularität und die weitere Zunahme der parallel ausführbaren Ereignisse steigen die Beschleunigungszahlen der balancierten Partitionierungsverfahren bereits bei gleichgewichtigem Aufwand auf Werte bis 9,9 bei 32 Prozessoren. Ein Einbruch der Leistung ist nicht zu beobachten. Die anderen Partitionierungsverfahren erreichen ebenfalls Speedups zwischen 6,3 und 7,3 bei 16 Prozessoren. Lediglich die prinzipiell stets eine einzige große Partition generierende Fan-in-Kegelpartitionierung kann natürlicherweise keine Beschleunigung erreichen.



a) Gleichgewicht b) hoher Berechnungsaufwand

Abbildung 8.13: Gegenüberstellung verschiedener Granularitäten für Modell s35932

Bei weiterer Erhöhung der Ereignisgranularität gelangen die Meßwerte in den Bereich der die Kommunikation berücksichtigenden CPA-Vorhersagen. Der Einbruch mit balancierten Partitionierungsmechanismen bleibt aus und ein Anwachsen der Beschleunigung auf 18,3 mit 32 Prozessoren wurde beobachtet. Die nichtausgewogenen Aufteilungen bleiben in ihrer Leistung unverändert.

Bezogen auf eine Gesamtlaufzeit der sequentiellen Simulation von 5392 bzw. 27278 Sekunden (resp. 1,5 und 7,5 Stunden) bedeutet die mögliche Reduktion der Simulationsdauer auf 10 bzw. 25 Minuten bereits erhebliche Zeitgewinne. Insgesamt zeigt sich bei der Modifikation der Granularität für eine reale parallele Implementierung, daß hierin ein großes Potential zum Erreichen signifikanter Beschleunigungen liegen kann. Allerdings nähern sich die grobgranularen Modelle durch die Reduktion der Bedeutung von Kommunikationskosten bei wachsender Ereignisdauer mehr und mehr einem datenparallelen Bearbeitungsmodell an, das in regelmäßigen Phasen für eine kurze Zeit Daten miteinander austauscht, um anschließend wieder in einen längeren Berechnungszyklus einzutreten. Aufgrund der Behandlung der Nachrichten innerhalb einen Kommunikationskoprozessors wird die Berechnung der Ereignisnachrichten während des Nachrichtenempfangs nicht einmal unterbrochen, so daß die Ähnlichkeit der beiden Algorithmenklassen noch verstärkt wird. Es kommt nicht mehr so sehr darauf an, wann die Nachrichten versandt werden, sondern es ist lediglich wichtig, daß entsprechende Informationen rechtzeitig vorliegen, damit die Simulatoren ungestört weiterarbeiten können.

Eine allgemeine Transformation von Schaltungen auf Gatterebene in ein grobgranulares Modell ist jedoch, wie bereits in Kapitel 7.4.2 erwähnt, nicht möglich, so daß die hier gefundenen Ergeb-

nisse möglicherweise für andere Applikationsklassen relevant sind, aber im Bereich der parallelen Simulation von Schaltungen nicht direkt verwendet werden können.

8.1.4 Einfluß des Cluster-Mappings

Aufgrund der Clusterung vieler einzelner Objekte, von denen jedes in den Grundverfahren der parallelen ereignisgesteuerten Simulation als eigenständiger LP modelliert wird, stellte sich die Frage, ob man durch das in DVSIM vorgenommene Verschmelzen der LPs und Nachrichtenkanäle nicht Parallelitätspotential verschenkt, da einzelne Objekte aufgrund zusätzlicher Randbedingungen durch andere Objekte blockieren.

Um die Größe der Cluster und dadurch möglicherweise künstlich konstruierte Wartebedingungen zu reduzieren, wurde die Clusteranzahl erhöht und mehrere Cluster auf einem Prozessor plaziert. Die Verteilung bleibt dabei weitgehend dem Kommunikationssystem PVM überlassen und erfolgt entsprechende einer Round-robin-Zuteilung nach den Partitionsnummern. Es gibt somit kein intelligentes Mapping, das z.B. Kommunikationsaufwand oder Topologie des Verbindungsnetzwerks berücksichtigt. Eine Integration in die Partitionierungsalgorithmen wäre jedoch durch eine zweite Partitionierungsstufe denkbar, die die Schnittkosten der Partitionen berücksichtigt und durch eine einfache Permutation der Partitionsnummern dafür sorgt, daß stark zusammenhängende Teilkomponenten denselben Prozessoren zugewiesen werden. Außerdem handelt es sich bei der Erhöhung der Anzahl von LPs um eine Parallelisierung mittels schwergewichtiger Prozesse, die miteinander nicht sonderlich effizient kommunizieren. Eine Betrachtung mit kleineren LPs, die als einzelne Threads innerhalb eines einzigen Prozesses ablaufen, wurde nicht durchgeführt.

Die Untersuchungen wurden am realen Modell s35932, das relativ zu den anderen Modellen bereits bei der Grundgranularität ein recht gutmütiges Verhalten zeigte, durchgeführt. Dazu wurden 8, 16, 32 und 64 Partitionen auf 8, 16 bzw. 32 Prozessoren verteilt so daß sich eine Partitionsdichte zwischen 1 und 8 pro Prozessor ergab. Aus diesen Parametern sollte man wenigstens Trends ableiten können, um die Notwendigkeit einer tiefergehenderen Untersuchung beurteilen zu können.

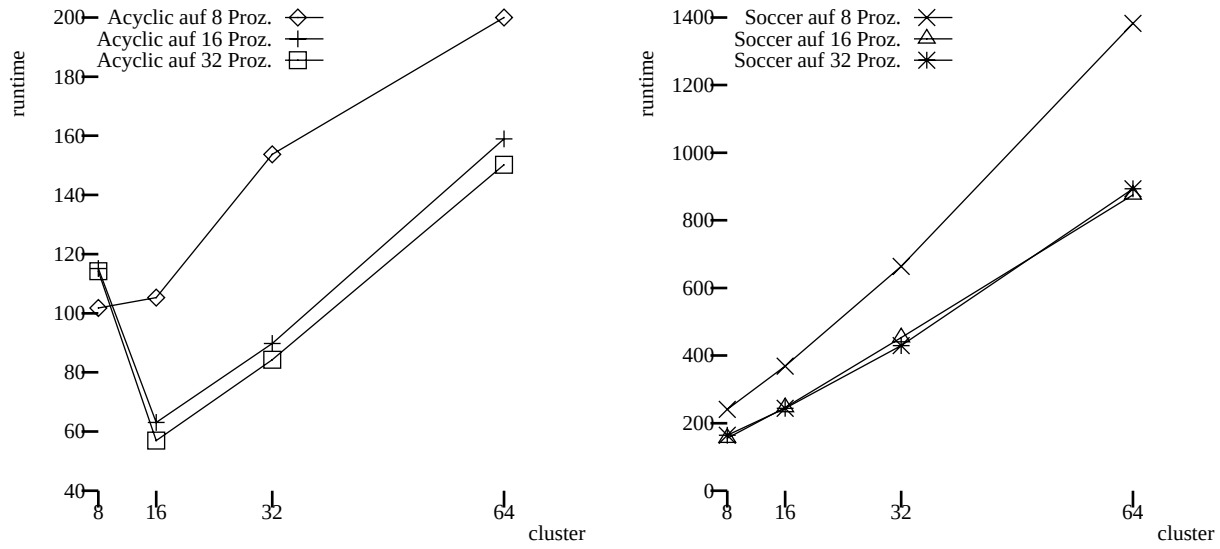


Abbildung 8.14: Einfluß der Clusteranzahl pro Prozessor, s35932

Trotz zunehmender Zuweisung mehrerer LPs pro Rechnerknoten ist keine Verbesserung der Laufzeiten aufgrund höherer Flexibilität beim Scheduling der LPs im Gegensatz zu den Versionen

mit maximal einem LP pro Prozessor zu bemerken (Abb. 8.14), die darauf schließen läßt, daß die Verwendung mehrerer kleiner Cluster zu einer effektiveren Bearbeitung führen könnte. Insgesamt bleibt die Anzahl der von einem Prozessor zu bearbeitenden Objekte konstant, da die Partitionen mit wachsender Clusterzahl kleiner werden[13]. Es ist nach den vorliegenden Ergebnissen sogar eher davon auszugehen, daß der Overhead zur Synchronisation der einzelnen LPs durch die fortschreitende Zerlegung wächst. Mit jeder Verdoppelung der Clusteranzahl erhöht sich die Laufzeit um das 0,3- bis 1,1-fache des Vorgängerwerts.

Die Nähe der beiden Kurven für 16 und 32 Prozessoren zeigt, daß bei der Reduktion von 32 auf 16 Prozessoren und der damit verbundenen Belastung der Prozessoren mit der doppelten Clusteranzahl wohl wegen der kleinen Partitionsgröße scheinbar noch freie lokale CPU-Kapazitäten aufgrund von Blockaden vorhanden sind, die zur Simulation der neu zugewiesenen Cluster ausgenutzt werden können.

Bei einer weiteren Halbierung der Prozessoranzahl auf 8 benötigen jedoch die einzelnen Rechnerknoten einen entsprechend hohen Rechenaufwand für die Simulation der bereits davor vorhandenen Cluster, so daß sich die Simulationsdauer insgesamt erhöht. Die bei acht Clustern und der azyklischen Partitionierung beobachtete schlechtere Laufzeit mit 16 bzw. 32 Prozessoren läßt sich nicht eindeutig erklären. Möglicherweise geht dieses Verhalten auf die Tatsache zurück, daß mehr Prozessoren vorhanden sind als von den LPs für das Mapping tatsächlich benötigt werden. Die Abbildung der Simulationsprozesse auf die Knoten der Parallelrechnerarchitektur könnte deshalb ungüstiger sein. Diese Mutmaßung hat jedoch recht spekulativen Charakter.

Inwieweit die Verwendung der schwergewichtigen Prozesse für die Leistungseinbrüche verantwortlich ist, konnte ebenfalls nicht eindeutig geklärt werden, da der Vergleich zur einer sich auf gemeinsamem Speicher synchronisierenden Threadarchitektur für die LPs fehlt. Es ist jedoch zu vermuten, das das Scheduling der schwergewichtigen Prozesse starken Aufwand generiert. Weiterhin tauschen die lokal auf demselben Prozessor angesiedelten LPs nach wie vor die Ereignisnachrichten über den PVM-Nachrichtenmechanismus aus, wodurch sich die steigenden Laufzeiten aufgrund des erhöhten Overheads bei wachsender Clusteranzahl ebenfalls verstärken, obwohl die lokale Kommunikation in der vorliegenden PVM-Implementierung gegenüber der Netzwerkkommunikation optimiert ist.

Daneben ist der Aufwand der Threadsynchronisation über gemeinsamen Speicher ebenfalls nicht zu unterschätzen. In einer frühen Arbeit wurde ein paralleler Warteschlangennetzsimulator mit einem LP pro Warteschlange nach diesem Mechanismus organisiert [MEI91a]. Die Koordination des korrekten Speicherzugriffs erforderte sorgfältige Planung der Datenstrukturen und generierte ebenfalls einen nichttrivialen Aufwand, um insbesondere Deadlocks zwischen den Threads zu vermeiden. Darüber hinaus wird auch bei Threads ein Scheduler benötigt, der zugegebenerweise wesentlich effizienter arbeitet.

Für eine allgemeine abschließende Aussage scheinen die Beobachtungen nicht ausreichend zu sein. Ein Nachweis der Leistungsfähigkeit bei Verwendung mehrerer unabhängiger LPs pro Rechnerknoten erfordert weitere Untersuchungen. Die Tatsache, daß mit erhöhter Granularität bzw. entsprechend balancierten Partitionsgrößen jedoch auch mit einer geclusterten Simulatorversion Beschleunigungen erreicht wurden, stimmt zuversichtlich, daß der hier verwendete Ansatz vernünftig und berechtigt ist, zumal sich der Organisationsaufwand der einzelnen Simulatoren dadurch erheblich reduziert.

[13] Dabei wird die Anzahl der Cluster entlang der einzelnen Kurven bei konstanter Prozessoranzahl bei jedem Meßpunkt von links nach rechts verdoppelt.

8.1.5 Beurteilung der konservativen Verfahren

Die Untersuchungen mit den konservativen Simulationsmethoden ergaben zunächst sehr enttäuschende Ergebnisse. Bei kleinen und mittleren Modellen ergaben sich für die Applikationsklasse der Schaltkreissimulation auf Gatterebene sogut wie keine Beschleunigungswerte gegenüber der sequentiellen Variante. Für das größere reale Modell s35932 konnten Beschleunigungswerte im Bereich von 25 Prozent des idealen linearen Speedups erreicht werden. Bei genauer Betrachtung stellte sich heraus, daß die Partitionierung bei den konservativen Verfahren zwei wichtige Aspekte berücksichtigen muß. Einerseits gilt es sicherzustellen, daß die Partitionen eine bestimmte Mindestgröße, die zwischen 3000 und 5000 Gattern liegt, nicht unterschreiten darf. Daraus ergibt sich auch ein Limit für die Anzahl der verwendeten Prozessoren für die Parallelsimulation in Abhängigkeit von den benutzten Modellen bzw. eine Mindestmodellgröße, ab der bei vorgegebener Rechenknotenanzahl mit Beschleunigungen gerechnet werden kann.

Andererseits spielen partitionsübergreifende Zyklen eine sehr nachteilige Rolle, da sie Deadlocks verursachen, die in der Regel zwar lokalen Charakter haben, aber effizient nur über globale Deadlockerkennungsalgorithmen entdeckt und dadurch erst entsprechend spät aufgelöst werden können. Partitionierungsalgorithmen, die Zyklen vermeiden, können hier mitunter gute Ergebnisse liefern, auch wenn sie gelegentlich mit der Zielsetzung des Balancierungskriteriums in Konflikt treten und dann sehr schlecht abschneiden.

Abstrahiert man jedoch von der konkreten Applikation und setzt eine wesentlich grobgranularere Ereignisbearbeitungsdauer voraus, so lassen sich mit den hier untersuchten Verfahren interessante Beschleunigungen erreichen. Der Aufwand, den Kommunikation in Rechensystemen mit verteiltem Speicher spielt, tritt in den Hintergrund. Die Ergebnisse decken sich dann teilweise mit den theoretischen Vorhersagen, die mittels Kritische-Pfad-Analyse erstellt werden können. Außerdem gestattet die CPA die effiziente Erkennung von Modellen mit großen Zyklen, die sich erwartungsgemäß schlecht mit konservativen Verfahren parallel simulieren lassen. Konservative Verfahren zur Vermeidung von Verklemmungen wurden nicht untersucht, da u.a. die Voraussetzung von Mindestgarantien für Simulationszeiten nicht gegeben ist.

8.2 Optimistische Verfahren

Die Untersuchungen der zweiten großen Klasse von Synchronisationsalgorithmen erfolgt anhand ihres bekanntesten Vertreters, des Time Warp [JES85a], und orientiert sich stark an der bereits im ersten Teil dieses Kapitels verwendeten Methodik. Die kritischen Faktoren werden herausgearbeitet und beschrieben. Die Betrachtungen stellen nach Möglichkeit sofort einen Bezug zu den konservativen Verfahren her, um eine gegenseitige Abgrenzung vornehmen zu können.

Die optimistischen Verfahren verfügen über eine weitaus höhere Anzahl von Freiheitsgraden als die konservativen Methoden. Als Ausgangspunkt unserer Untersuchungen der optimistischen Protokolle wurden zunächst Messungen mit dem Lazy-cancellation-Verfahren (LC) durchgeführt, da diesem in der Literatur das höchste Beschleunigungspotential zugeschrieben wird. Als Einstiegspunkt wurden wie auch im konservativen Fall die real auf dem GC/PP-Rechnersystem vorhandene Basisgranularität für Nachrichten und Ereignisse verwendet.

8.2.1 Grundverfahren

Die folgenden Experimente basieren auf dem Time Warp mit der LC-Erweiterung von Gafni [GAF88a]. Insbesondere ist festzustellen, daß hierbei jeder LP mit ungebremster Geschwindigkeit

arbeitet und es prinzipiell keine Limitierungen bezüglich der bearbeitbaren Ereignisse gibt[14]. Die Berechnung der GVT wird vom Kontrollprozessor (LP mit Partitionsnummer 0) gesteuert und im Abstand von mindestens 50 Ereignissen durch diesen initiiert. Bei einer Anzahl von mindestens einigen Tausend bearbeiteten Ereignissen pro Prozessor ist damit eine permanante Neuberechnung der GVT-Approximation garantiert.

8.2.1.1 Reale Schaltungen

Die Tabellen 8.13 – 8.15 zeigen die Laufzeiten der Experimente für die verwendeten realen Schaltungen. Mit dem kleinen Modell s1196 lassen sich auch unter dem Time Warp keine Beschleunigungen erreichen. Allerdings fallen die Slowdowns bei K/L und Soccer hier nicht so dramatisch aus wie im konservativen Fall.

Anzahl Part.	Simulationsdauer			
	RR	azykl.	K/L	Soccer
1		5,14		
8	31,10	2,74	171,88	3,75
16	83,72	2,46	122,32	22,63

Tabelle 8.13: Simulationsdauer mit optimistischem Verfahren, s1196

Bei den Experimenten mit s13207 ergibt sich zunächst wiederum ein Abfallen der Laufzeiten auf einen Minimalwert bis zu einer gewissen Prozessoranzahl. Anschließend zeigen die Resultate leider wieder eine steigende Tendenz für weiter abnehmender Partitionsgröße. Interessanterweise machen hier jedoch die balancierten Kegelverfahren eine Ausnahme. Sie erreichen unter Time Warp bereits für dieses mittlere Modell Beschleunigungen bis zu 6 auf 24 Knoten. Azyklische und unbalancierte Kegelverfahren stagnieren aufgrund der unvorteilhaften Aufteilung der Partitionen bei Speedup-Werten um "1".

Anzahl Part.	Simulationsdauer								
	RR	azykl.	Fan-in Cones		Fan-out Cones		K/L	Soccer	String
			unbal.	bal.	unbal.	bal.			
1				143,99	144,02	144,07			
2		82,73	99,46	81,52	86,07	75,60	no mem.	81,42	9520,32
4	no mem.	61,18	104,16	41,98	88,65	39,95	no mem.	49,87	1684,73
8	no mem.	62,34	102,99	24,58	no mem.	34,82	no mem.	26,52	756,02
12	no mem.	60,60	103,55	16,83	no mem.	15,88	no mem.	108,13	496,67
16	no mem.	63,02	103,05	13,90	no mem.	19,17	no mem.	144,59	338,66
24		60,11	108,55	11,70	89,33	10,64	no mem.	19440,88	no mem.
32		62,36	103,65	17,53	86,57	21,87	no mem.	291,16	433,65
48						32,13			

Tabelle 8.14: Simulationsdauer mit optimistischem Verfahren, s13207

Die Verläufe der Beschleunigungswerte fallen jedoch insgesamt nicht so harmonisch aus wie bei den konservativen Verfahren. Vielmehr treten plötzliche und eigentlich unerwartete Sprünge auf,

[14]Der Optimismus des Time Warp wird durch verschiedene Umgebungsvariablen gesteuert. In diesem Fall wird der unlimitierte Optimismus durch die Festlegung der minimalen Fenstergröße auf den Wert ∞ eingestellt.

bei denen die Simulationszeiten vollständig aus dem Rahmen der umgebenden Werte fallen. Ein Beispiel dafür ist die Soccerpartitionierung mit 24 Knoten in Modell s13207. Eine klare Ursache ist hier nicht erkennbar. Die Anzahl der von jedem Prozessor zu bearbeitenden Ereignisse, die nicht zurückgestzt werden (commited events) ist für alle LPs einigermaßen ausgewogen. Es fällt jedoch auf, daß in den Fällen mit erhöhter Laufzeit einzelne Prozessoren eine hohe Anzahl von Rollbacks ausführen müssen und zwischen zwei Rollbacks sehr viele Ereignisse wiederholt ausgeführt werden (Coasting-forward, [JEF85a]). Diese "Aufholphase" im Anschluß an einen Rollback ist also in den betreffenden Experimenten extrem teuer.

Anzahl Part.	Simulationsdauer							
	azykl.	Fan-in Cones		Fan-out Cones		K/L	Soccer	String
		unbal.	bal.	unbal.	bal.			
1	462,41							
8	115,26						181,87	
12	195,78	489,77	117,71	90,27	85,73	1209,70	292,73	no mem.
16	67,56	488,69	109,89	90,69	82,14	no mem.	121,66	no mem.
24	226,51	498,53	105,65	no mem.	191,53	no mem.	no mem.	no mem.
32	158,91	493,11	100,87	no mem.	98,00	no mem.	no mem.	no mem.

Tabelle 8.15: Simulationsdauer mit optimistischem Verfahren, s35932

Modell	Part.-alg..	Anzahl Part.	mittlerer Aufwand der Rollbacks (in %)	maximaler Aufwand der Rollbacks (in %)
s1196	k	8	21.5	47.2
s13207	a	4	3.0	13.1
s13207	a	8	4.2	13.2
s13207	c	16	1.4	9.7
s13207	D	24	0.6	1.0
s13207	D	32	1.8	16.9
s13207	s	16	5.7	25.4
s13207	s	24	1.9	26.6
s13207	s	32	3.0	60.3
s13207	S	2	26.3	40.3
s13207	S	4	23.2	33.9
s13207	S	8	20.8	32.4
s35932	a	12	17.1	25.4
s35932	a	16	7.4	15.3
s35932	a	24	12.5	24.7
s35932	s	12	24.4	36.1

Tabelle 8.16: Anteil der Rollbacks an der Simulationszeit

Einen Ausreißer ins andere Extrem stellt der Simulationslauf mit 16 Knoten und azyklischer Partitionierung für Schaltung s35932 dar. Bei dieser Konfiguration ist das Verhältnis zwischen wiederholt ausgeführten Ereignissen und Gesamtzahl der Ereignisse entsprechend günstig, wodurch sich das bessere Abschneiden des Experiments erklären läßt. Der Anteil für die alleinige Ausführung der Rollbacks bezogen auf die Gesamtsimulationsdauer liegt hier bei maximal 15,3 Prozent. Die beiden

benachbarten Messungen benötigen dagegen etwa 1/4 der Rechenzeit für die Korrekturmaßnahmen. Darin sind die Kosten der Coasting-forward-Phase jedoch noch nicht enthalten.

Tabelle 8.16 zeigt für die betrachteten Modelle den Aufwand für die Durchführung der Rollbacks und stellt ihn der Simulationsdauer gegenüber. Der Overhead durch wiederholte Ereignisausführung läßt sich nur schwer ermitteln, da sich Rollbacks überlagern und so nicht definiert ist, wann die Berechnung die von Jefferson definierte Coasting-forward-Phase wieder verläßt und effektiv "neue" Ereignisse ausführt. Verhalten sich diese in etwa proportional zum Rollbackaufwand[15], so erhält man einen nicht unerheblichen Overhead.

Weiterhin fällt auf, daß der Zusatzaufwand für die verteilte Simulation mit lediglich einem Prozessor wesentlich höher als beim konservativen Verfahren ist. Hier schlägt vermutlich der (wenn auch geringe) Aufwand für die inkrementelle Zustandssicherung zu, die, auch wenn sie bei der Einprozessorversion nicht nötig ist, dennoch ausgeführt wird. Der Vergleich der Meßwerte gegen diesen Wert anstelle der Laufzeit des sequentiellen Simulators lieferte wiederum überhöhte Beschleunigungszahlen.

Ein problematischer Aspekt der Untersuchungen des Time Warp stellt die Tatsache dar, daß etliche Simulationsläufe mit Speichermangel abbrachen. Diese Beobachtung läßt sich ebenfalls nicht an bestimmten Partitionierungsverfahren, Schaltungen oder Prozessoranzahlen eindeutig festmachen.

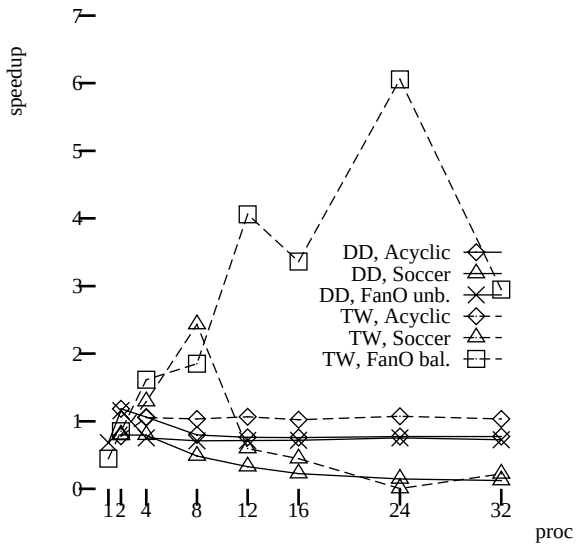


Abbildung 8.15: Direkter Vergleich zwischen Deadlock Detection und Time Warp, s13207

Die Daten der Tabellen 8.13 – 8.15 spiegeln keine eindeutige Tendenz wider, die die Favorisierung eines bestimmten Partitionierungsverfahrens für einen beliebigen optimistischen oder konservativen Synchronisationsmechanismus rechtfertigt. Statt dessen zeigen sich mit dem in der Literatur oft bevorzugten Time Warp teilweise sogar schlechtere Beschleunigungswerte als sie mit dem konservativen Grundverfahren erzielt werden konnten (Abbildungen 8.15 und 8.16).

In Modell s13207 überrascht die balancierte Fan-out-Kegelpartitionierung mit ihren Ergebnissen. Es werden Beschleunigungen bis 6 erreicht und auch ein Absinken der Werte für zunehmende Partitionsanzahl ist nicht direkt zu erkennen. Lediglich die bereits angesprochenen Schwankungen

[15] Diese Annahme ist plausibel, da bei einem Rollback im Prinzip die Ereignisliste rückwärts durchlaufen wird bis der Zeitpunkt des aufgetretenen Fehlers erreicht ist. Dabei wird der Zustand der Objekte wiederhergestellt und es existieren kaum zu löschende Ereignisse. Bei der nachfolgenden Wiederausführung der Ereignisse werden quasi dieselben Events nochmals bearbeitet.

treten bei 16 Knoten auf. Ab 32 Prozessoren brechen die Leisungen wieder ein. Es handelt sich dabei wohl nicht nur um einen Ausreißer, da eine Kontrollmessung mit 48 Partitionen eine weitere Verringerung der Geschwindigkeit ergab.

Die Ergebnisse für s35932 können nicht an diejenigen der Schaltung s13207 anknüpfen. Die Resultate bei Time Warp erreichen zwar auch Speedups, jedoch liegen diese unterhalb der mit konservativen Verfahren erreichbaren Spitzenwerte. Weiterhin unterliegen sie starken Schwankungen. Eine Vorhersage darüber, welche Beschleunigungen mit welcher Prozessoranzahl zu erwarten sind, wird dadurch extrem schwierig. Lediglich die unbefriedigende Angabe weit auseinanderklaffender unterer und oberer Schranken ist hier möglich.

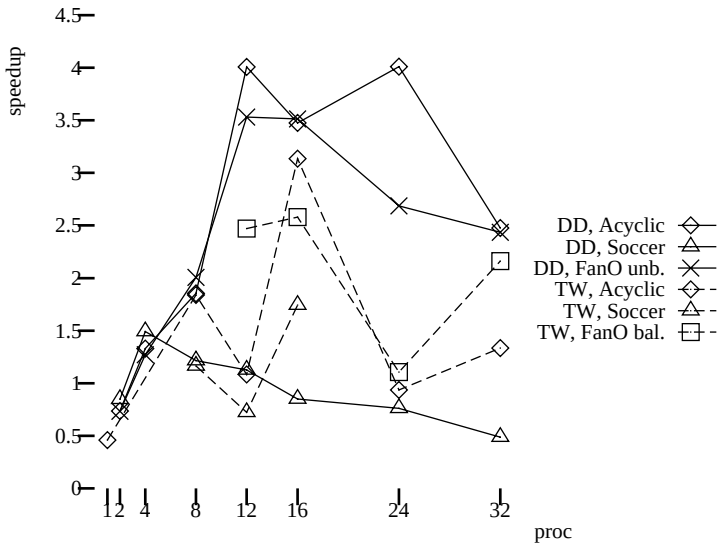


Abbildung 8.16: Direkter Vergleich zwischen Deadlock Detection und Time Warp, s35932

Es scheint jedoch so zu sein, daß im Time Warp die Partitionierungsverfahren, die in etwa gleich große Partitionen berechnen, relativ gut im Vergleich zu den anderen Verfahren abschneiden, da sie eine übermäßige Entkoppelung der einzelnen Simulatoren vermeiden. Durch die Notwendigkeit mehr oder weniger häufig Antinachrichten zu versenden, wird erreicht, daß die einzelnen Komponenten nicht zu sehr in der Simulationszeit auseinandertriften. Somit wird die Tiefe und der Aufwand der Rollbacks relativ gering gehalten. Bei der zyklenfreien Variante verursacht gerade das Fehlen von Rückkopplungen einen Überoptimismus der Simulation, der durch eine hohe Rollbackrate bestraft wird.

8.2.1.2 Synthetische Schaltungen

Die kleinen synthetischen Modelle (ca. 70 Gatter) erreichen bei einem absoluten Slowdown von etwa 80 Prozent für eine Vielzahl von Partitionierungen beim Time Warp bessere absolute Laufzeiten als die konservativen Pendants. Diese Tatsache ändert jedoch nichts daran, daß sich auch Time Warp für kleine Schaltungen nicht gewinnbringend einsetzen läßt. Teilweise liegen die Laufzeiten auch höher als für die konservativen Verfahren.

Bei den mittleren synthetischen Benchmarks (ca. 10000 Gatter) zeigten sich mit dem Basisverfahren des Time Warp ebenfalls schlechtere Spitzenwerte als bei den konservativen Methoden. Weiterhin sind bereits die Laufzeiten der Einprozessorversion wiederum höher als bei den konservativen Verfahren, so daß auch hier zunächst unfaire relative Speedups berichtet werden könnten,

wenn man nicht sorgfältig arbeitet.

Dennoch zeigte sich ein homogeneres Verhalten bezüglich der mit unterschiedlichen Partitionierungsverfahren erreichbaren mittleren Beschleunigung. Die Kluft zwischen den Meßwerten für unterschiedlichen Aufteilungsmethoden ist weitaus geringer und die meisten der beobachteten Simulationsläufe auf 8 oder 16 Prozessoren weisen Beschleunigungen zwischen 1,5 und 3 auf.

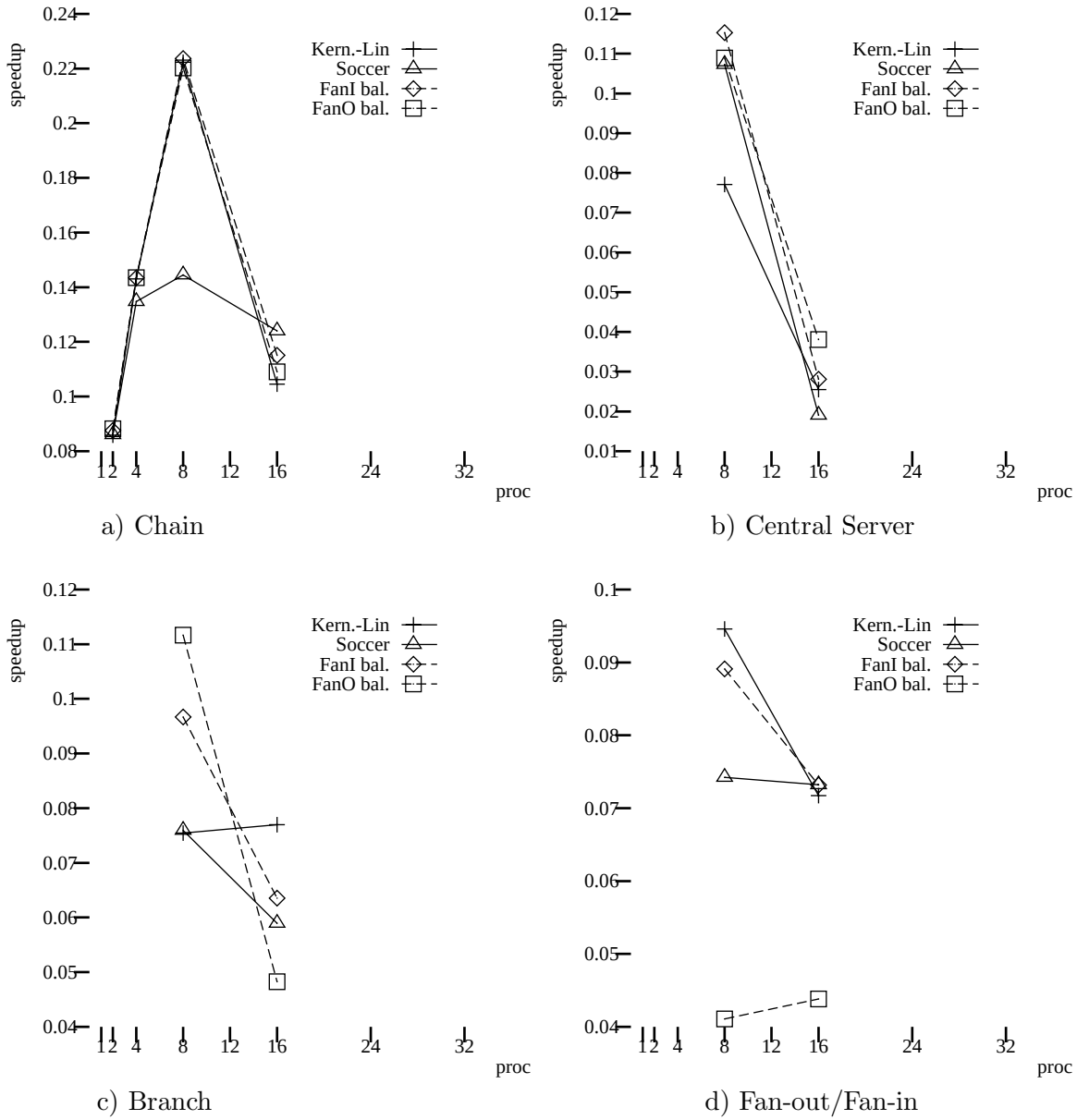


Abbildung 8.17: Speedup Time Warp, inv64*

Bei der Untersuchung der größeren Schaltung inv100000_chain werden auch mit wachsender Prozessoranzahl nicht die Laufzeitreduktionen erzielt, wie man sie bei Deadlock-detection beobachtet (Abb. 8.19). Die Speedups im Time Warp erreichen nur etwa 60 Prozent davon. Insbesondere erbringt das beim konservativen Deadlockvermeidungsalgorithmus gute azyklische Verfahren im Ti-

me Warp[16] nur mittelmäßige Resultate. Auch für dieses Modell wurden wieder Simulationsabbrüche aufgrund von Speichermangel beobachtet (bei Soccer mit 8 Prozessoren).

Die Feststellung, die für die realen Schaltungen getroffen wurde, daß sich keines der Synchronisationsverfahren eindeutig durch seine Leistung hervorhebt, bestätigt sich also auch bei den synthetischen Benchmarks.

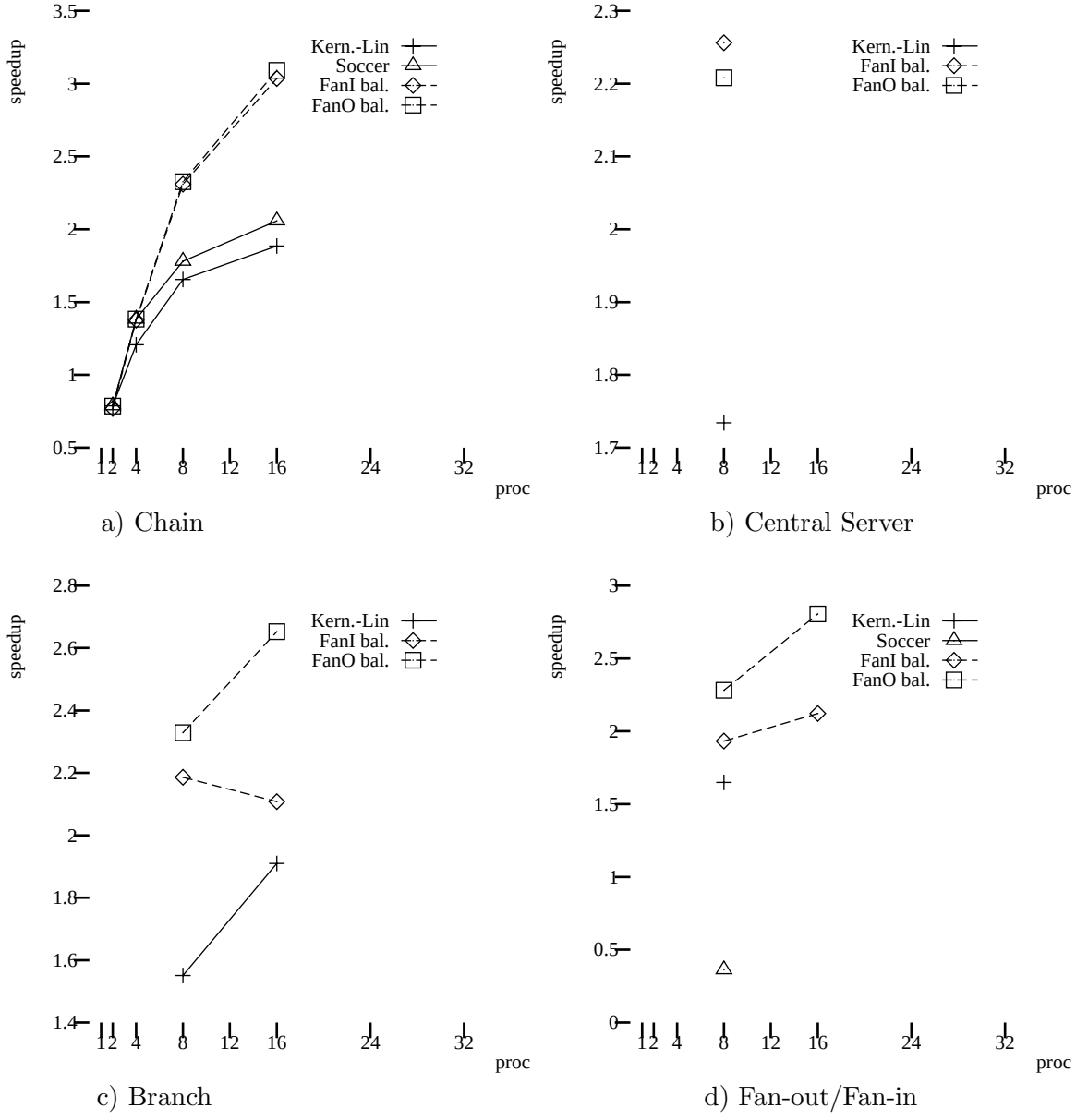


Abbildung 8.18: Speedup Time Warp, inv10000*

Um die Ursache für das teilweise schlechte Abschneiden des Time Warps zu ermitteln, werden nachfolgend weiterführende Untersuchungen durchgeführt. Zum einen wurde die Granularität erhöht, was ja bei den konservativen Verfahren bereits eine starke Leistungssteigerung erzielen konnte. Andererseits wurde aber auch versucht unter Beibehaltung der Basisgranularität für die

[16] Hier repräsentiert durch die balancierten Kegelpartitionierungen

vorliegende Klasse der Schaltkreissimulation auf Gatterebene weitere Verbesserungen zu erreichen.

Die untersuchten Optionen beinhalten Time Warp nach dem aggressiven Verfahren, um die überschnelle Ausbreitung der feingranularen Ereignisse zu bremsen und die Limitierung des Optimismus durch Fenstertechniken. Die Beobachtungen werden in den folgenden Abschnitten beschrieben und analysiert.

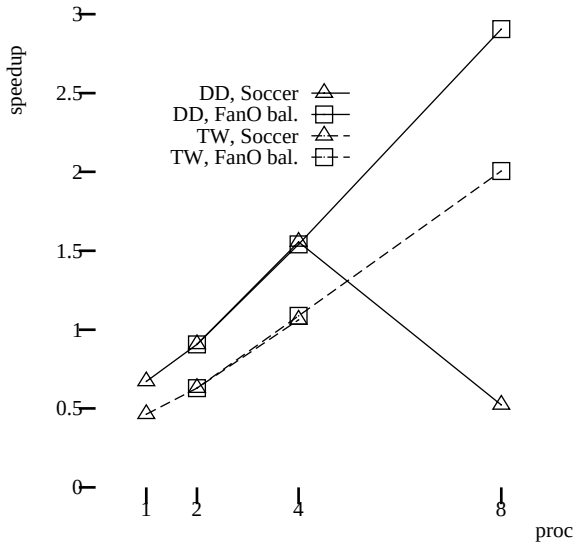


Abbildung 8.19: Vergleich DD und TW, inv100000_chain

8.2.2 Erhöhung der Granularität

Es wurden bei folgenden Ergebnissen ebenfalls die eingeführten Standardverhältnisse bezüglich Kommunikations- und Berechnungsanteil verwendet (ausgewogenes Verhältnis, 1:1, und Überwiegen der Berechnung, 5:1).

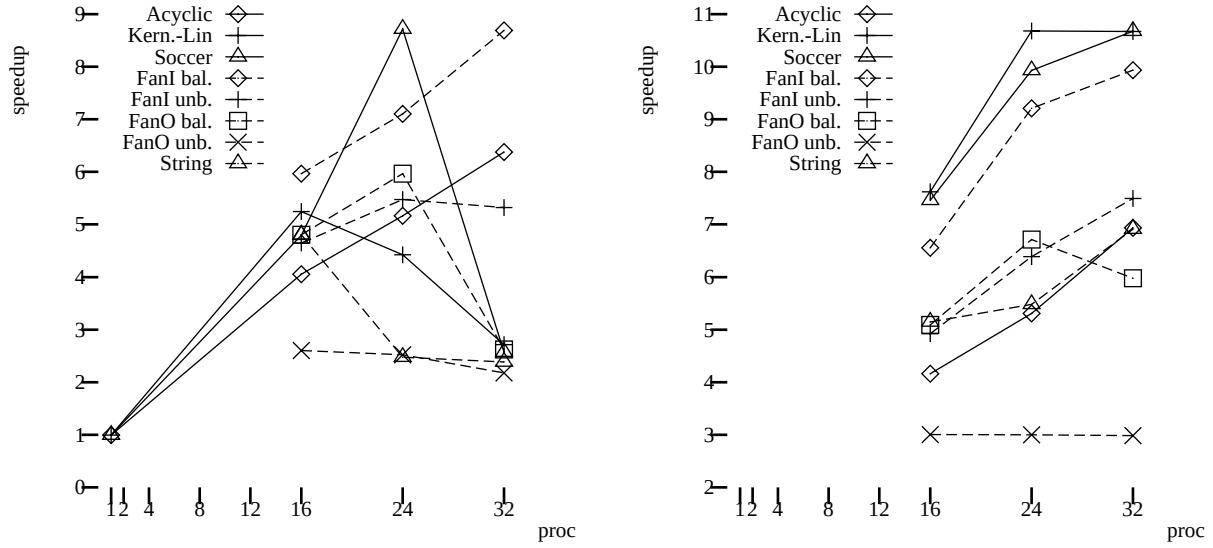
Die Ereignisgranularität scheint im Time Warp noch stärker als beim konservativen Verfahren Einfluß auf die Leistungsfähigkeit des parallelen Algorithmus zu nehmen. Bereits im kleinen Modell s1196 werden für viele Partitionierungsverfahren steigende Beschleunigungswerte mit Spitzen bei 8,7 auf 24 Prozessoren für das Soccerverfahren registriert. Danach brechen die Beschleunigungswerte wieder drastisch ein. Eine Ausnahme stellen hier die azyklische und balancierte Fan-in-Kegelpartitionierung dar, die ein Speedup-Wachstum auf Maximalwerte 6,4 und 8,7 mit 32 Prozessoren erreichen.

Der Mehraufwand für die verteilten Synchronisationsverfahren in der Einprozessorvariante ist aufgrund der erhöhten Granularität vernachlässigbar klein geworden, so daß sich in Abb. 8.20a ein Speedupwert von "1" dafür ergibt.

Überwiegt der Berechnungs- gegenüber dem Kommunikationsaufwand, so zeigen sich bis auf K/L und Soccer keine wesentlichen Verbesserungen. Lediglich diese beiden Verfahren erreichen weiteres Wachstum der Beschleunigung auf den Wert 10,7 mit 32 Prozessoren und übertreffen damit sogar die Ergebnisse der balancierten Fan-in-Kegelpartitionierung. Die Minimierung der Schnittkosten für die Kommunikationskanäle durch diese beiden Verfahren scheint sich hier auszuwirken.

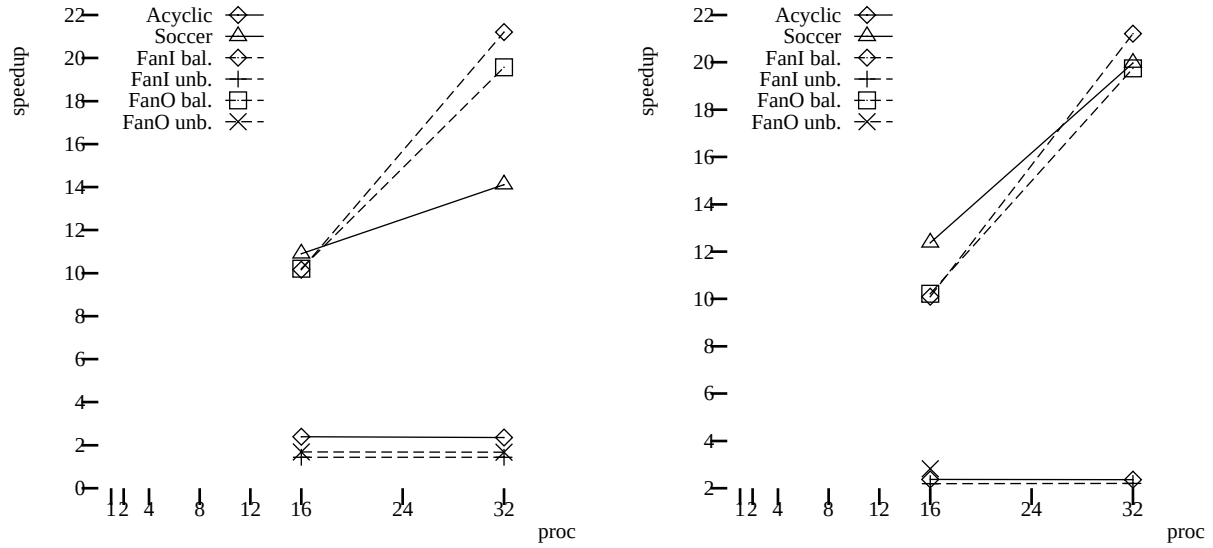
Die Betrachtungen für Modell s13207 verstärken diese Trends weiter. Bereits mit ausgewogenem Aufwand ergeben sich Beschleunigungswerte von bis zu 21,2 mit den balancierten Kegelverfahren. Die unbalancierten Kegelmethoden und die ohnehin ungleichgewichtige azyklische Partitionierung

verhalten sich ähnlich wie im Fall der Basisgranularität mit einem konstanten Beschleunigungspotential auf dem niedrigen Niveau um den Wert "2". Mit wachsender Granularität gleichen sich die guten Verfahren in ihren Ergebnissen lediglich noch gegenseitig an, ohne weitere Speedups zu erreichen.



a) Gleichgewicht b) hoher Berechnungsaufwand

Abbildung 8.20: Erhöhung der Granularität bei Time Warp, s1196

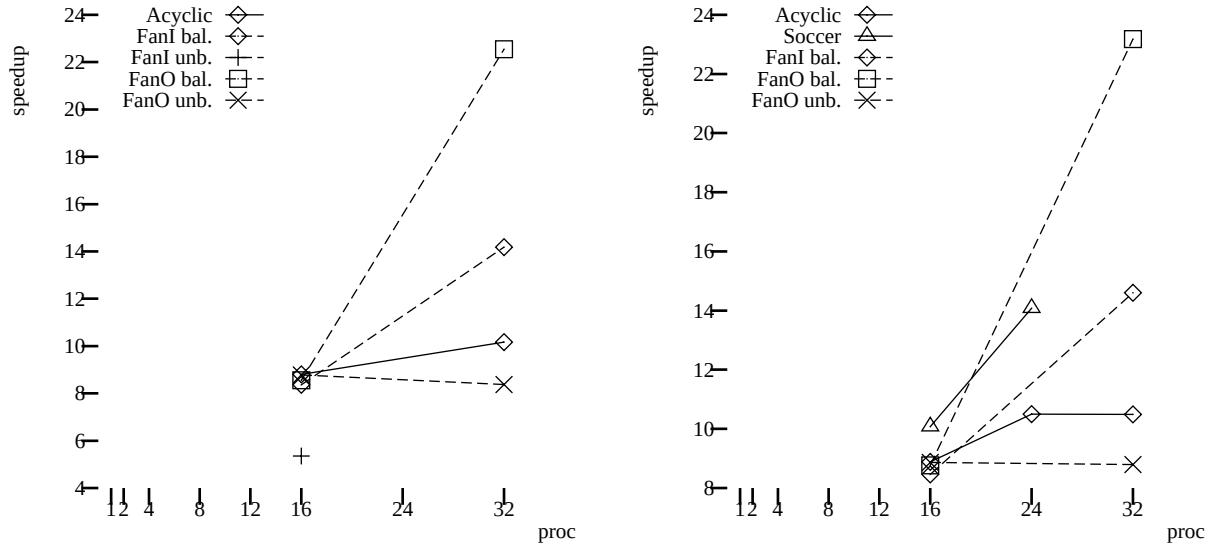


a) Gleichgewicht b) hoher Berechnungsaufwand

Abbildung 8.21: Erhöhung der Granularität bei Time Warp, s13207

Auch mit Schaltung s35932 zeigen sich Beschleunigungswerte, die nahezu 75 Prozent der optimalen linearen Beschleunigung erreichen. Dabei unterscheiden sich die Ergebnisse für ausgewogene Granularität und ein Überwiegen der Berechnungen kaum noch. Aufgrund der Vermeidung von Blockaden und stetem Ausführen von Ereignissen sofort nach deren Verfügbarkeit existieren über

den gesamten Simulationszeitraum hinweg genügend parallel ausführbare Ereignisse. Verwendet man dazu eine größenbalancierte Aufteilung der Schaltung, kann dieser Vorteil effektiv genutzt werden.



a) Gleichgewicht b) hoher Berechnungsaufwand

Abbildung 8.22: Erhöhung der Granularität bei Time Warp, s35932

Die Bedeutung der Rollbacks am Gesamtaufwand nimmt ab, da ihre Ausführung immer noch denselben Zeitaufwand wie im Grundverfahren besitzt. Bei etwa 25 Prozent Overhead bei Einsatz der Grundgranularität sinkt dieser Anteil bei einer Vervielfachung der Granularität rein rechnerisch auf weniger als 1 Prozent. Daneben wurde auch beobachtet, daß die Anzahl der Rollbacks und deren Tiefe (gemessen in der Anzahl der zurückgesetzten Ereignisse) mit steigender Granularität abnimmt und dadurch ein weiterer Beitrag zur Kostenreduktion geleistet wird.

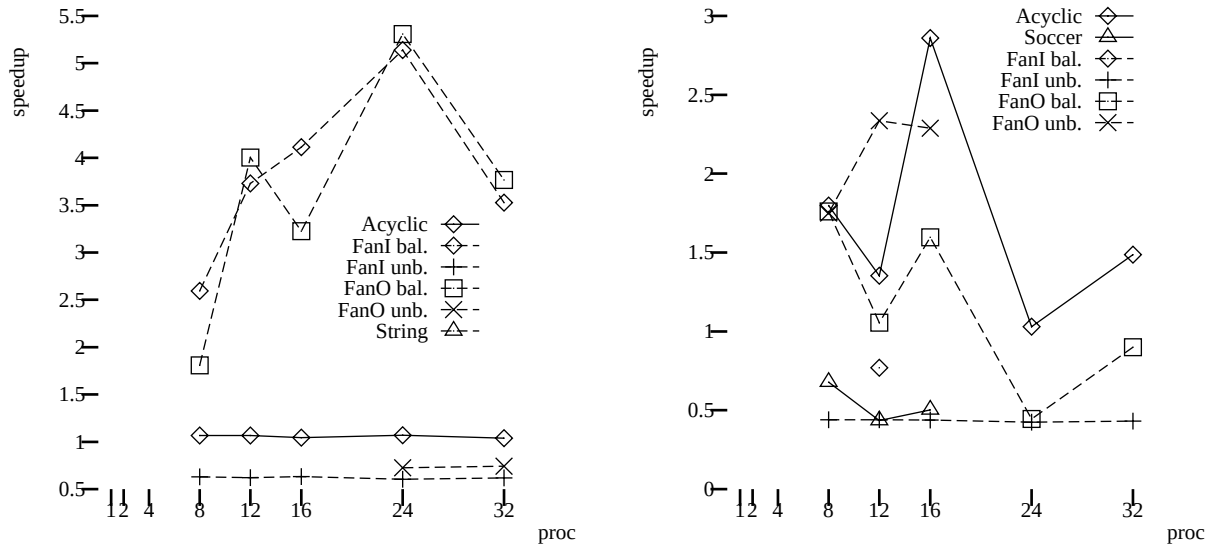
Insgesamt verhält sich der Time Warp als optimistisches Verfahren für steigende Granularität günstiger als die konservativen Methoden. Es wurden sehr interessante Beschleunigungswerte festgestellt, die nahe an den Idealwerten der linearen Beschleunigung in Abhängigkeit von der Prozessorenanzahl liegt. Leider ist jedoch auch hier zu bemerken, daß diese Aussage ebenso wie für konservative Synchronisation nur für grobgranulare Applikationen gilt. Durch den asynchronen Charakter der Zeitforschaltung wird der mögliche Parallelismus gleichzeitig ausführbarer Ereignisse prinzipiell erweitert. Es gilt jedoch zu prüfen, wie hoch dieser Gewinn ist und ob dieselben oder ähnliche Werte für diese Applikationsklassen nicht auch mit synchronen Verfahren erreichbar sind, die weitaus weniger Implementationswissen und -aufwand voraussetzen[17]. Für die bei der parallelen Logiksimulation verfügbare Granularität sind die optimistischen Verfahren ebenfalls nur eingeschränkt nutzbar, um Beschleunigungen der Simulationsläufe zu erzielen.

8.2.3 Aggressive-cancellation

Da bei Verwendung der Basisgranularität die Rollbackkosten einen nicht unerheblichen Anteil haben, wurden die Simulationen für die Modelle s13207 und s35932 unter Einsatz des Time Warp mit aggressivem Widerruf möglicherweise falscher Ereignisse (AC) durchgeführt. Dadurch können

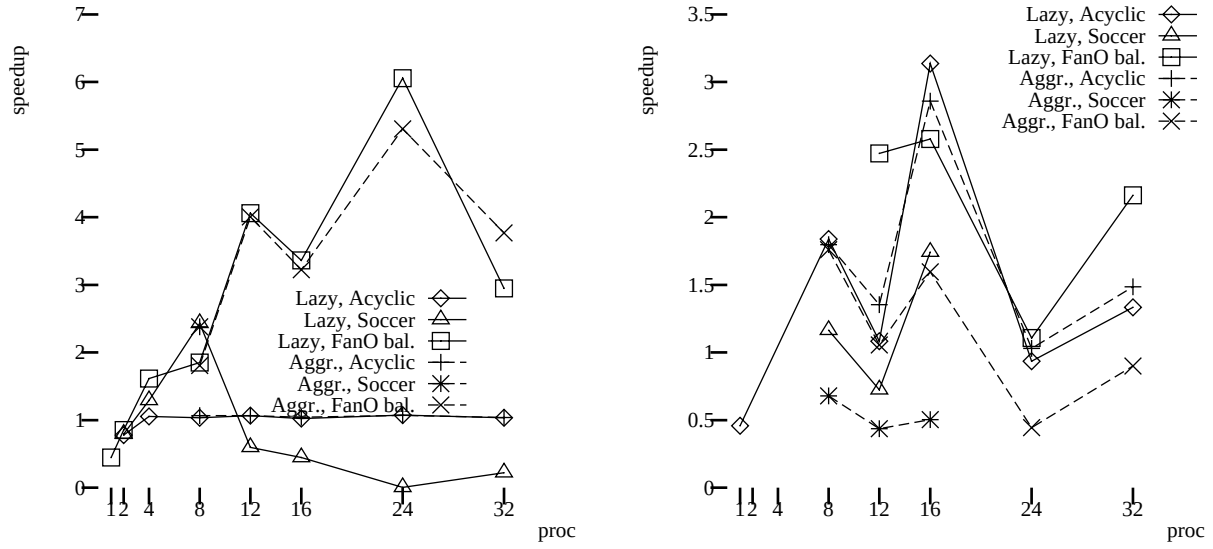
[17] Soulé untersuchte diese These und fand auf Shared-memory-Rechnern die synchrone Simulation auf Gatterebene dem konservativen Chandy-Misra-Bryant-Verfahren überlegen [SOU92a].

prinzipiell tiefere Rollbacks durch frühzeitiges Rückgängigmachen extern generierter falscher Ereignisse vermieden werden. Diesen Vorteil erkauft man sich eventuell durch eine höhere Anzahl zu versendender Antinachrichten. Die Graphen in Abbildung 8.23 zeigen für die Modelle s13207 und s35932 die Beschleunigungswerte unter Verwendung der aggressiven Korrekturvariante.



a) s13207 b) s35932

Abbildung 8.23: Aggressive-cancellation mit Grundgranularität



a) s13207 b) s35932

Abbildung 8.24: Vergleich zwischen Aggressive- und Lazy-cancellation

Die Ergebnisse bei Modell s13207 stimmen für beide Verfahren fast vollständig überein. Es läßt sich keinerlei Aussage über eine bessere Vorgehensweise treffen. Erst die größere Schaltung s35932 offenbart leichte Vorteile des Lazy-cancellation-Verfahrens. Bei Soccer und balancierter Fan-out-Kegelpartitionierung zeigen sich Leistungseinbrüche von 50 Prozent im Vergleich zu LC. Die Anzahl

der Rollbacks steigt entgegen der Erwartung, daß die Rücksetzungen durch frühzeitiges Propagieren fehlerhafter Berechnungen beschränkt werden könnten, sogar an. Vermutlich werden durch das sofortige Versenden der Antinachrichten viele sekundäre Rollbacks ausgelöst, die eigentlich gar nicht notwendig wären, weil die meisten Nachrichten anschließend wieder identisch generiert werden.

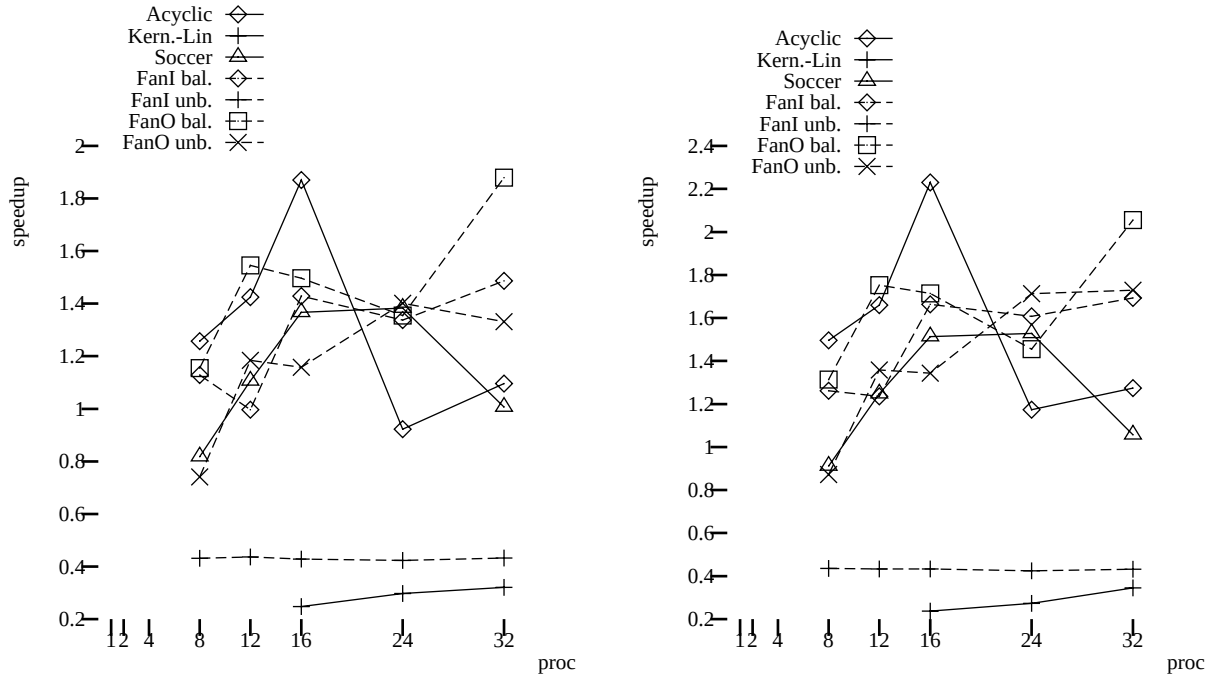
Die azyklische Partitionierung kann jedoch auch in der aggressiven Variante weiterhin mit den Ergebnissen von Lazy-cancellation konkurrieren. Bei 12, 24 und 32 Prozessoren werden sogar kürzere Laufzeiten als mit LC erzielt. Die Anzahl der Rollbacks liegt bei AC zwar höher als mit LC aber die Tiefe der Rollbacks wurde demgegenüber leicht reduziert, so daß sich in diesen Fällen wirklich eine Verringerung des Rollbackaufwands ergibt.

Da sich offenbar Lazy-cancellation bezüglich der betrachteten Schaltungen etwas gutmütiger verhält als das aggressive Gegenstück, basieren die nachfolgenden Untersuchungen zur Einschränkung des unbegrenzten Optimismus auf LC.

8.2.4 Limitierung des Optimismus

Eine in der Literatur oft betrachtete Optimierungsmethode liegt in der Definition eines Simulationszeitfensters, das den Optimismus des Time Warp limitiert. Verschiedene bisher gemachte Beobachtungen zeigen, daß diese Vorgehensweise durchaus Berechtigung hat. Balancierte Partitionierungen wiesen für den Time Warp eine meist bessere Leistung auf als stark entkoppelte wie die unbalancierten Kegelmethoden oder azyklische Aufteilungen der Schaltkreise.

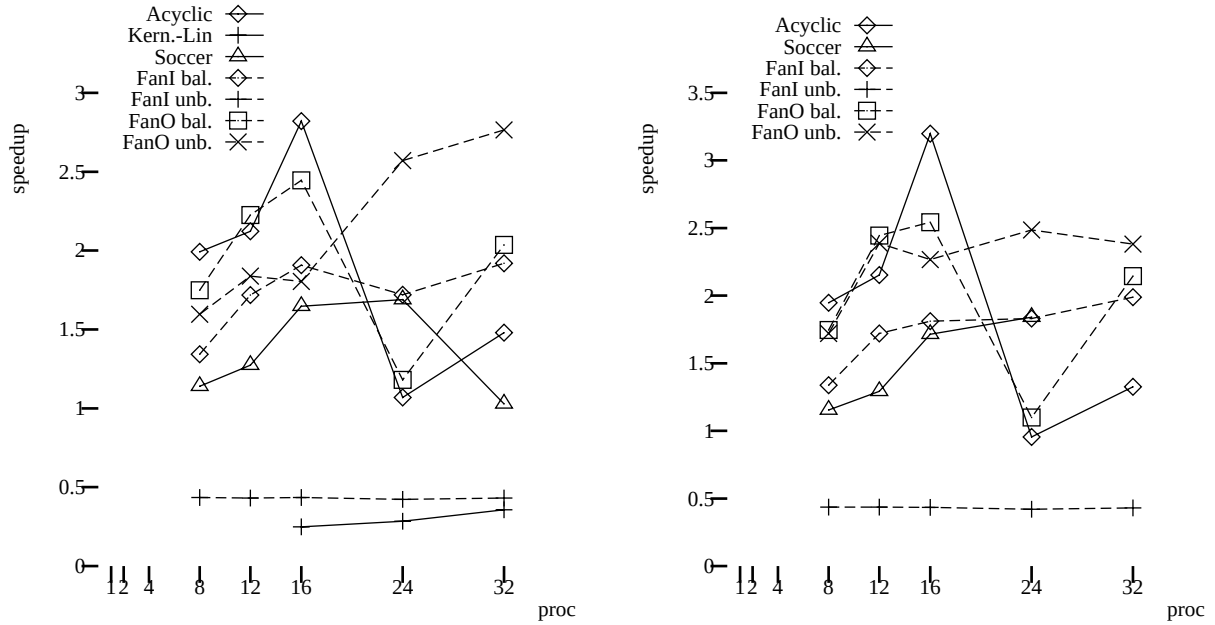
In den folgenden Untersuchungen wurde zunächst empirisch versucht, eine optimale statische Fenstergröße zu ermitteln. Diese Vorgehensweise dient zunächst der Orientierung, in welchen Bereichen Verbesserungen zu erwarten sind. Dabei liegt die Fenstergröße aller Teilsimulatoren auf einem konstanten und für alle Prozessoren identischen Wert.



a) Größe 500 b) Größe 1000

Abbildung 8.25: Limitierung des Optimismus durch Fenstertechniken, s35932

Die Abbildungen 8.25 und 8.26 enthalten für die festen Fenstergrößen 500, 1000, 5000 und 25000[18] die Beschleunigungskurven beim größten betrachteten Modell s35932 mit verschiedenen Partitionierungsverfahren. Es ergeben sich hinsichtlich der Qualität der Graphen kaum Unterschiede zum Basisverfahren ohne den Einsatz von Fenstertechniken. Einzige beobachtet Ausnahme stellt die unbalancierte Fan-out-Kegelpartitionierung dar, die bei einer Fenstergröße von 5000 ihre Speedupwerte auf 2,8 mit 32 Prozessoren steigert und damit im Bereich des Spitzenwerts für die azyklische Partitionierung liegt.



a) Größe 5000 b) Größe 25000

Abbildung 8.26: Limitierung des Optimismus durch Fenstertechniken, s35932

Jedoch besitzt die Limitierung des Optimismus einen negativen Einfluß auf die Gesamtlaufzeit der Simulation. Je kleiner die Fenstergröße gewählt wird, desto länger dauerte die Bearbeitung. Gegenüber der unbeschränkten Version benötigt die limitierte Variante teilweise bis zur doppelten Laufzeit. Mit den sehr restriktiven Fenstern in Abb. 8.25a und b liegen fast alle beobachteten Speedups zwischen den nicht gerade überzeugenden Werten "1" und "2".

Positiv zu bemerken ist dabei, daß eine Vielzahl von Simulationsläufen, die zuvor mit Speichermangel abgebrochen wurden, bei Verwendung fast aller Fenstergrößen erfolgreich zu Ende geführt werden konnten. Die statische Festlegung von Fenstergrößen scheint also eher ein adäquates Mittel zur Kontrolle des Speicherbedarfs im Time Warp als für die Minimierung der Laufzeiten zu sein.

Anhand der hier gewonnenen Informationen zeigt sich eine Größenbeschränkung von 500 – 1000 Simulationszeiteinheiten durchaus als förderlich für verschiedene Partitionierungsansätze. Die selbstrelativen Beschleunigungen für wachsende Prozessorzahlen steigen hier im Gegensatz zu unlimitiertem Optimismus an (z.B. Kernighan/Lin). Dennoch ist kein absoluter Speedup (> 1) im Vergleich zur sequentiellen Simulation zu beobachten. Bei den kleineren Modellen wurden ähnliche Tendenzen beobachtet.

[18] Die Fenstergröße wird in 1/10 ns Simulationszeit angegeben und entspricht somit der Basiszeiteinheit der VHDL-Simulation. Bei einer Gesamtdauer von 6,4 μs gibt das größte Fenster knapp den halben Simulationszeitraum zur Bearbeitung frei.

Bessere Ergebnisse werden allerdings von einem adaptiven dynamischen Verfahren erwartet, das mit Initial- und Mindestgröße des Fensters, sowie einem definierten Aktualisierungszeitrahmen startet. Im Anschluß an die statischen Untersuchungen wird deshalb die Ausdehnung des Fensters dynamisch von jedem Simulator zur Laufzeit dem aktuellen Verhalten angepaßt, um die Rollbackanzahl und -tiefe zu minimieren. Die Beschreibung der Adaptionsregeln erfolgte bereits in Abschnitt 6.5.1.2.

Als Startgröße des Fensters wurde der Wert 1000 für die adaptive Variante verwendet, um zu Beginn der Simulation eine gewisse Drosselung des Optimismus vorzugeben. Die Anzahl der Aktualisierungen sollte sich an der Rollbackhäufigkeit des Modells orientieren, um einerseits nicht zu häufige Anpassungen als Reaktion auf kurzfristige Schwankungen durchzuführen, aber andererseits auch langfristig negative Trends rechtzeitig korrigieren zu können.

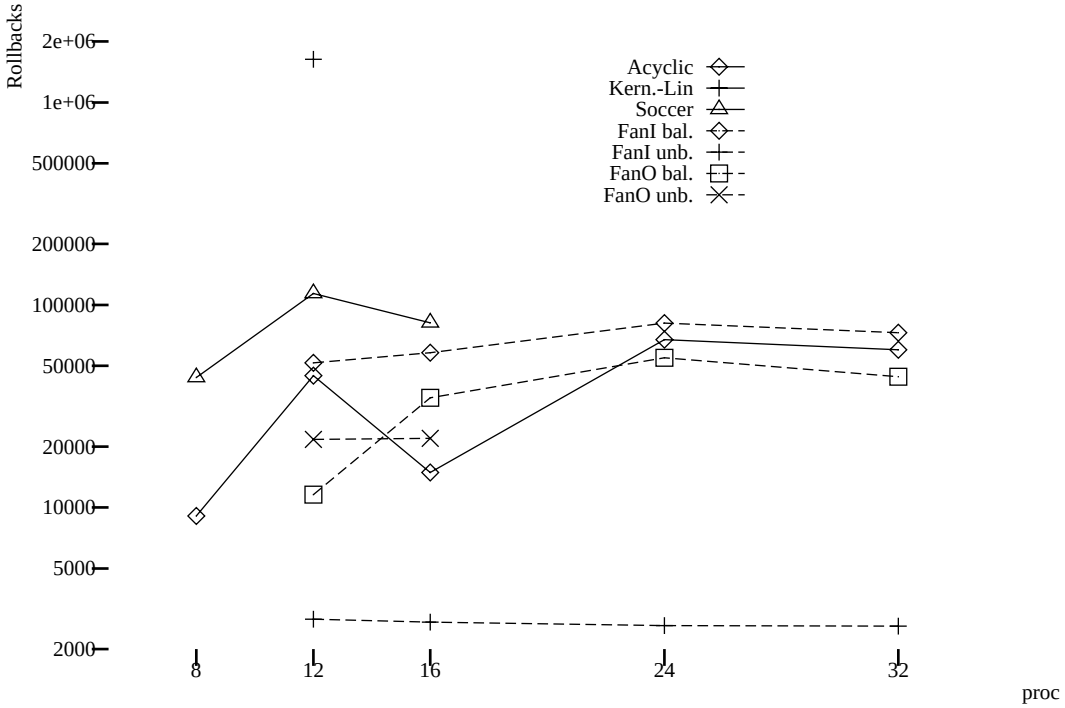
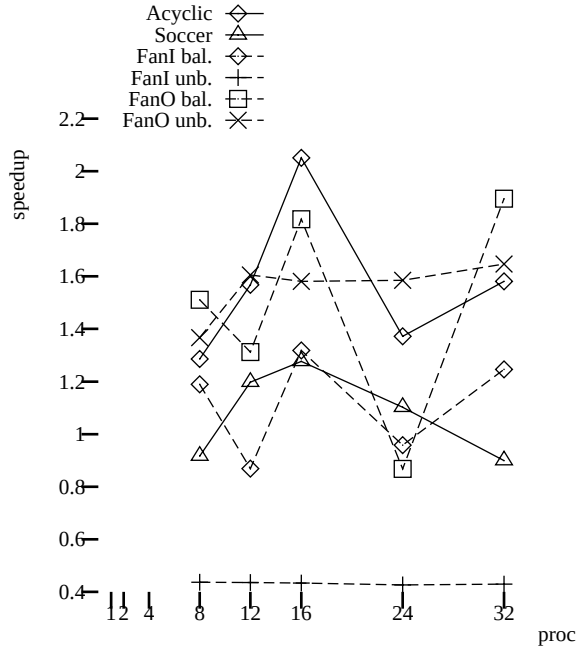


Abbildung 8.27: Anzahl der Rollbacks, s35932

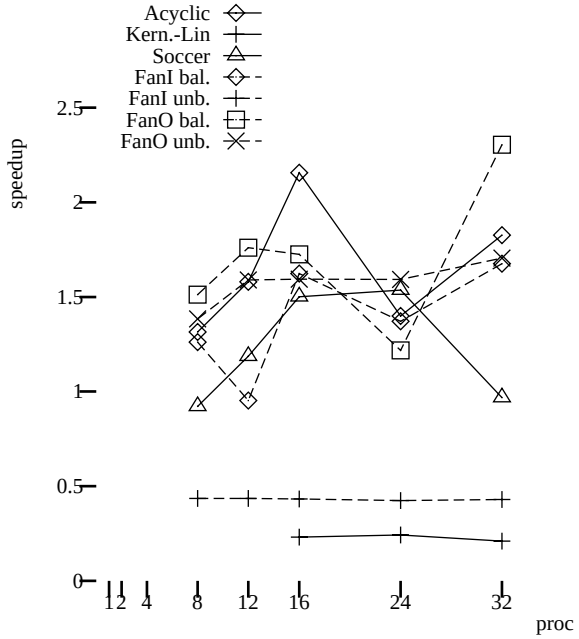
Die mittlere Rollbackhäufigkeit der einzelnen LPs für verschiedene Partitionierungen wird in Abbildung 8.27 dargestellt. Um eine hinreichend häufige Adaption der Fenstergröße zu gewährleisten, wurde die Aktualisierungsrate der Fenstergröße aus dem Bereich zwischen 100 und 5000 Rollbacks mit den konkreten Werten 100, 1000 und 5000 gewählt, wobei für Simulatoren mit ohnehin niedriger Rollbackrate[19] unter Umständen gar keine Anpassung in Richtung eines kleineren Fensters erfolgen muß. Treten keine Rollbacks zwischen zwei GVT-Aktualisierungen auf, so wird die Fenstergröße selbstverständlich verdoppelt. Die Ergebnisse dieser Untersuchungen werden in

[19]Die Abbildung gibt die mittlere Rollbackanzahl über alle Knoten an. Die Werte streuen jedoch sehr stark, und es gibt auch LPs mit extrem wenig oder gar keinen Rollbacks.

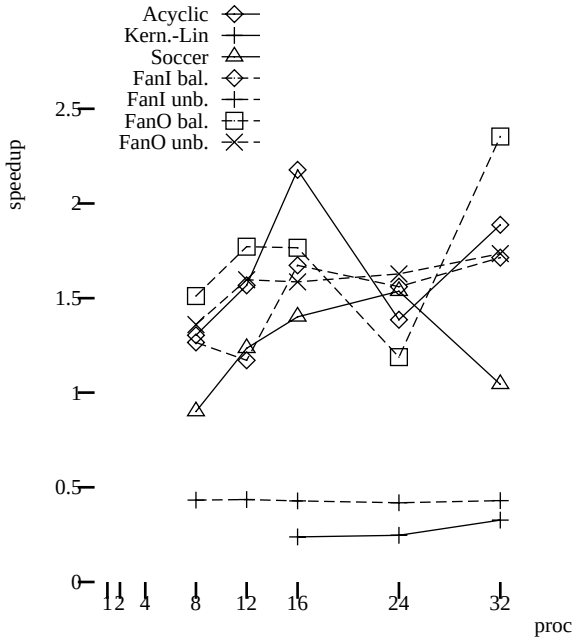
Abbildung 8.28 wiedergegeben.



a) Aktualisierungsrate, 100 Rollbacks



b) Aktualisierungsrate, 1000 Rollbacks



c) Aktualisierungsrate, 5000 Rollbacks

Abbildung 8.28: Dynamische Anpassung der Fenstergröße, s35932, Startfenstergröße 1000

Auch bei adaptiver Anpassung der Fenstergröße ergeben sich keine wesentlichen Veränderungen gegenüber dem statischen Verfahren. Eine zu häufige Anpassung der Fenstergröße scheint der Leistungsfähigkeit des Time Warp sogar eher abträglich. Außerdem besitzt die die Wahl der initialen

Fenstergröße wohl einen stärkeren Einfluß auf die Gesamtleistung als erwartet. Bei einer initialen Fenstergröße von 5000 Zeiteinheiten steigt die Leistung des dynamischen Verfahrens im Vergleich zu kleineren Startfenstern teilweise noch leicht an, jedoch ohne die Ergebnisse mit statischer Fenstergröße zu erreichen.

Zusammenfassend läßt sich feststellen, daß verschiedene in der Literatur genannten Einflußfaktoren wie Synchronisationsverfahren und Fenstertechniken einen eher untergeordneten Einfluß auf die Leistungsfähigkeit paralleler optimistischer Simulationssysteme spielen. Die Parallelisierbarkeit bei Applikationen mit niedriger Ereignisgranularität ist relativ gering und erreicht teilweise nicht die Werte, die mit konservativen Verfahren erzielt wurden. Die optimistischen Varianten Lazy-cancellation und Aggressive-cancellation weichen nur wenig in ihren Ergebnissen voneinander ab. Bei Verwendung von Fensterverfahren wurde in DVSIM sogar eine Dämpfung der Leistung beobachtet. Ein wesentlicher Vorteil der Fenstertechniken scheint dagegen die Kontrolle des Speicherbedarfs zu sein, so daß Simulationen nicht aufgrund fehlender Kapazitäten vorzeitig abbrechen.

Aji, Palaniswamy und Wilsey [APW93a] untersuchten als Einflußgrößen auf das Laufzeitverhalten des Time Warp den Einsatz von relativ kleinen Zeitfenstern (nach unserer Einstufung in der Größenordnung 0 - 100), Cancellation-Strategien, Checkpointing-Häufigkeit und Optimierung des Rollback-Mechanismus durch Kennzeichnung zustandsloser Elemente, die beim Rollback nicht beachtet werden müssen. Die Kombination kleiner Fenster mit LC und periodischen Sicherungen im Abstand von 3 – 5 Ereignissen lieferten die besten Ergebnisse im Vergleich zum reinen Time Warp. Der Einfluß der Fenstertechnik nimmt auch in diesen Untersuchungen einen eher untergeordneten Rang ein und bedeutet nur stellenweise marginale Verbesserungen von wenigen Prozent.

Die Speedup-Kurven, die bei optimistischen Verfahren ermittelt wurden, zeigen gelegentlich ein sprunghaftes Verhalten. In [LIL90a] wird angeführt, daß die Hinzunahme weiterer Prozessoren bei Time Warp nur dann sinnvoll ist, wenn genügend Parallelität im Modell vorhanden ist. Andernfalls kann die Speedupkurve starken Schwankungen unterliegen (sog. S-Phänomen). Es werden Bereiche in Abhängigkeit von der statistisch erwarteten Korrektheit der optimistisch ausgeführten Ereignisse angegeben, in denen die Erweiterung der Prozessoranzahl für STF-Scheduling (Shortest-timestamp-first) Sinn oder keinen Sinn macht. Anhand von Messungen mit verschiedenen Modellen untermauerten Lin und Lazowska diese These.

Diese Beobachtungen könnten Ursache der beobachteten Schwankungen bei den Leistungskurven von DVSIM sein. Die Clusterung mehrerer LPs stellt quasi ein STF-Scheduling aller lokalen Ereignisse dar. Durch die Verwendung statischer Partitionierungsverfahren, die das exakte Verhalten zur Laufzeit nicht berücksichtigen, ist eine Abschätzung der real entstehenden Last a priori nicht möglich. Bei den konservativen Verfahren bremst der langsamste Prozessor die anderen, da er ggf. Deadlocks verursacht oder Garantien fehlen. Dadurch wird ein zu starkes Abweichen der Leistung einzelner LPs verhindert. Die optimistischen Verfahren dagegen generieren durch ungleiche Auslastung eventuell nicht vorhersagbaren Overhead in Form von Rollbacks und Antinachrichten, was zu den genannten Schwankungen führt. Eine Überprüfung mit den theoretischen Ergebnissen von Lin wurde nicht durchgeführt, da es mit den verwendeten Schaltungen schwierig ist, eine statistische Verteilung der korrekt ausgeführten Ereignisanzahlen über den Simulationszeitraum zu erstellen.

8.2.5 Weitere Aspekte

Neben diesen grundlegenden Betrachtungen der optimistischen Verfahren gibt es noch einige Gesichtspunkte, die nicht detailiert untersucht wurden, oder bei denen keine befriedigenden Lösungen gefunden wurden.

- Die anfängliche Entwicklung von DVSIM auf einem Intel iPSC/860-Hypercube bereitete insbesondere bei Time Warp aufgrund des limitierten Hauptspeichers von 8 bzw. 16 MB pro Knoten erhebliche Probleme. Aus diesem Grund wurde eine Implementierung einer globalen Speicherkontrolle mittels des Cancelback-Verfahren angedacht und auch prototypisch vorgenommen. Allerdings läßt sich der von Jefferson [JEF90a] vorgeschlagene Algorithmus auf verteiltem Speicher wegen fehlender global aktueller und atomar zugreifbarer Zustandsinformation sowie realer Laufzeiten von Nachrichten nicht einfach realisieren. Insbesondere traten erhebliche Probleme mit der verteilten Berechnung der GVT auf, da auch zurückgeschickte Nachrichten bei der GVT-Ermittlung berücksichtigt werden müssen. Dies ist mit einem echt verteilten Algorithmus ohne Einfrieren des Systems nur mit sehr viel Aufwand machbar. Mit dem Wechsel auf den GC/PP wurden die Restriktionen aufgrund der Speicherlimitierung ohnehin etwas gelockert. Dennoch ist nicht zu vernachlässigen, daß das Problem weiterhin besteht und auch auf dem besser ausgebauten Rechnersystem massiv wieder auftrat.

  Das und Fujimoto beschreiben in [DAF93a] eine ähnliche Vorgehensweise und definieren die GVT als Minimum aus lokalen Simulationszeiten (LVT) und Zeitstempeln aller Nachrichten, die gerade verschickt oder aufgrund eines Cancelbacks zurückgesandt werden ($min\{LVT + t(msg) + t(resent - msg)\}$). Allerdings stellt jedes Zurückweisen einer Nachricht auch bei ihnen einen starken Eingriff in die parallele Ablaufstruktur der LPs dar, da sich alle Prozessoren global bei der Berechnung der GVT synchronisieren müssen. Dies kommt dem von uns beobachteten Einfrieren des Systems nahe, das mit hohen Kosten verbunden ist. Aus diesem Grund wurde der Ansatz nicht weiter verfolgt.

- Das Checkpointing der VHDL-Prozesse für den Time Warp ist noch nicht vollständig implementiert. Insbesondere würden bei einer vollständigen Sprachunterstützung mit Funktionsaufrufen, die unser Prototyp nicht anbietet, Aspekte der Zustandsspeicherung der Stack- und Aufrufstruktur sowie der Variablenhistorie benötigt, um im Falle eines Rollbacks auch die VHDL-Prozesse in den korrekten Zustand vor Eintreten der fehlerhaften Berechnung zurückzuversetzen.

- Untersuchungen zum erwarteten Speicherbedarf des Time Warp wurden nicht durchgeführt. Die in der Literatur oft verwendeten Verfahren beziehen sich auf statistische Annahmen, die mit unseren realen Schaltungen schlecht zu modellieren sind. Andererseits werden die Betrachtungen dort auch weitgehend für zwei oder nur wenig mehr Prozessoren durchgeführt. Meist wurde ein optimierter Speicherbedarf mit höheren Laufzeiten erkauft. Da die im Basisverfahren beobachteten Ergebnisse bereits recht unbefriedigend sind, ist Verbesserung durch solche Verfahren für größere Prozessorzahlen nicht zu erwarten.

8.3 Oracle-log

Zum Oracle-log-Verfahren wurden nur einige wenige Messungen mit den synthetischen Schaltungen inv10000_chain und inv10000_cycle durchgeführt, die ein lineare Kette von Invertern bzw. einen geschlossenen Ring darstellen. Als Partitionierungsverfahren kamen die Soccer-Partitionierung und für inv10000_chain auch die azyklische Variante zum Einsatz. Für die realen Modelle wurden Maximalwerte von 85 Prozent der Gesamtlaufzeit für Blockaden und Deadlockauflösung beobachtet (Abb. 8.9). Bei der Simulation der hier verwendeten künstlichen Schaltungen blockieren einzelne LPs sogar fast vollständig (Tabelle 8.18).

Das Oracle-log-Verfahren erreicht bei inv10000_chain eine drastische Reduktion der Laufzeit für die Soccer-Partitionierung, so daß mit 8 Prozessoren eine echte Beschleunigung von 2,5 erzielt werden kann. Das Basisverfahren lief dagegen etwa 11-mal langsamer als die sequentielle Version. Deadlocks treten nicht mehr auf und die maximalen Blockadeanteile, die durch das Warten auf möglicherweise fehlende Nachrichten verursacht werden, sinken auf 2/3 der Simulationsdauer. Diese Blockaden sind unvermeidlich, da hier tatsächlich auf den Empfang ausstehender Nachrichten gewartet wird.

Für die Schaltung inv10000_cycle wird immerhin auch noch eine um den Faktor 5,4 verbesserte Laufzeit der Soccer-Variante unter Oracle-log registriert. Die prozentualen Anteile der Blockaden haben sich hier jedoch gegenüber der konservativen Basisversion erhöht. Da jedoch die Gesamtlaufzeit insgesamt wesentlich stärker abnimmt, erhält man durch die Verwendung des zusätzlichen Wissens aus den Orakeldaten auch hier für Soccer auf 8 Prozessoren eine reale Verbesserung. Als maximale Blockadedauer wurde somit bei inv10000_cycle mit Oracle-log 202 statt 820 Sekunden beobachtet. Dies entspricht einer effektiven Reduktion der Blockaden um mehr als 75 Prozent. Für Modell inv10000_chain verringert sich die absolute maximale Wartezeit auf weniger als 3 Prozent des ursprünglichen Wertes.

Modell	Part.	Anz.	Laufzeit	
			Grundverfahren	Oracle-log
inv10000_chain	a	4	70,11	68,61
inv10000_chain	a	8	49,69	48,58
inv10000_chain	a	16	100,27	40,58
inv10000_chain	s	4	70,00	68,70
inv10000_chain	s	8	1213,74	51,68
inv10000_cycle	s	8	1746,33	322,83

Tabelle 8.17: Gegenüberstellung der Laufzeiten von DD und Oracle-log

Modell	Partitionierung		Blockaden (in %)		Deadlock-
	Alg.	Knoten	max.	avg.	anteil (in %)
inv10000_chain	a	8	7	4	0
inv10000_chain	s	8	97	47	27
inv10000_cycle	s	8	47	28	18
oracle, inv10000_chain	a	4	46	32	0
oracle, inv10000_chain	a	8	53	42	0
oracle, inv10000_chain	a	16	72	63	0
oracle, inv10000_chain	s	4	46	33	0
oracle, inv10000_chain	s	8	63	48	0
oracle, inv10000_cycle	s	8	63	45	0

Tabelle 8.18: Blockade- und Deadlockauflösungsanteil, künstliche Benchmarks

Absolut betrachtet konnte jedoch auch mit dem Oracle-log-Verfahren kein relevanter Speedup erzielt werden. Daneben ergaben sich aber weitere interessante Perspektiven durch die Beobachtungen des azyklischen Partitionierungsverfahrens. Bei vielen Messungen fiel die Gesamtlaufzeit durch den Einsatz von Oracle-log kaum merklich. Statt dessen stieg der Anteil der Blockadezeiträume drastisch an. Diese Tatsache belegt, daß die parallele Simulation prinzipiell auch in kürzerer Zeit

durchführbar sein sollte als hier beobachtet, da beide Verfahren identische Berechnungsergebnisse generieren. Durch die Existenz der Orakeldaten erhalten die LPs implizite Garantien in Form von Lookahead, d.h. Ereignisse können bereits bis zum Zeitpunkt der nächsten erwarteten Nachricht ausgeführt werden, während im konservativen Verfahren erst auf das Eintreffen dieser Nachricht gewartet werden muß. Beim Basisverfahren ist also eine gesteigerte Blockadedauer zu erwarten. Die Ergebnisse zeigen nun aber gerade ein umgekehrtes Verhalten.

Die Ursache dafür liegt bei den betrachteten pipelineartigen Modellen darin, daß bei Oracle-log bereits früher mit der Ereignisabarbeitung begonnen werden kann und keine ausreichende Ereignisanzahl für die einzelnen Simulationsschritte vorhanden ist (siehe auch Kap. 7.4.1). Der entsprechende Simulator ist mit der Abarbeitung der Ereignisse in einer Vielzahl der beobachteten Fälle bereits fertig, ehe eine neue Ereignisnachricht eines anderen LPs eintrifft. Aus diesem Grund blockiert die Simulation nahezu vor jedem Empfang der nächsten Nachricht. Diese These wird durch zwei weitere Beobachtungen gestützt. Erstens treten bei allen LPs unter Oracle-log fast identische Blockadedauern auf, da sie in etwa dieselbe Ereignisanzahl bearbeiten und dieselbe Verarbeitungsstruktur (eine sequentielle Teilkette der Schaltung) besitzen. Zweitens sinken die Blockadezeiten bei Reduktion der Stimulilänge proportional dazu.

Im Basisverfahren warten die einzelnen LPs zwar zu Beginn länger auf die Erlaubnis zum Start der ersten Ereignisabarbeitung[20]. Danach sind aber stets genügend Ereignisse vorhanden, um die Simulatoren nicht blockieren zu müssen. Außerdem wurde festgestellt, daß in den wenigsten Fällen auf das Eintreffen der nächsten Ereignisnachricht gewartet werden mußte.

Die angegeben kürzeren Blockadedauern des Basisverfahrens sind jedoch nicht ganz fair gegenüber den Werten der Oracle-log-Methode. Bei Oracle-log beinhaltet die Blockadedauer auch die Zeiten zum Entgegennehmen der während der Blockaden aufgelaufenen neuen Nachrichten. Da diese Aufgabe zum Simulationszyklus gehört, ist der LP während dieser Zeiten eigentlich nicht mehr blockiert. Eine exakte Messung hätte aber wesentlich mehr Meßpunkte zur Erfassung der Zeiten im Simulator erfordert. Die realen Zeiten, in denen ein konservatives Verfahren auf eine Erhöhung der Garantien warten muß, dürften sich also zwischen diesen beiden Extremen bewegen. Nimmt die Ereignisanzahl dagegen stark zu, so sollten die gemessenen Blockadezeiträume relativ exakt sein.

Die dargestellten Blockadedauern beinhalten dabei auch nicht die Zeiten, in denen einzelne LPs, die bereits ihre Simulation abgeschlossen haben, auf die Terminierung der restlichen Simulatoren warten. Durch eine Angleichung der Terminierungszeitpunkte ließe sich ggf. weitere Laufzeit einsparen.

8.4 Gesamtkosten der verteilten Simulation

Neben den reinen Simulationskosten werden nun wie bereits in Kapitel 7.3 die Gesamtkosten der verteilten Simulation betrachtet. Dazu werden die diejenigen Parameterkombinationen verwendet, die sich in den bisherigen Untersuchungen als optimal herausgestellt haben. Über die Summe der Einzelkosten werden dann die Minima der mit den parallelen Algorithmen benötigten Laufzeiten für jedes Modell gebildet. Die einzelnen Faktoren wurden in Teil I dieser Arbeit spezifiziert. In den Tabellen 8.19 und 8.20 werden die beobachteten Werte für die konservative und die optimistische Version angegeben. Dabei zeigten sich für beide Klassen von Synchronisationsmethoden die balancierten Kegelpartitionierungen den anderen Aufteilungsverfahren klar überlegen.

Der effektiv erreichbare Speedup schmilzt somit bei Berücksichtigung der zusätzlichen Kosten

[20]Sie beginnt frühestens nach dem Eintreffen der zweiten Ereignisnachricht, wenn keine initialen Ereignisse vorhanden waren.

weiter zusammen. Die Endresultate sind mehr als enttäuschend und liegen bei den optimalen Konfigurationen unterhalb von "2". Die Vergleichswerte der sequentiellen Variante beinhalten ebenfalls den dort zusätzlich anfallenden Aufwand.

Modell	Simulationsdauer								
	Part.	Anz.	t_{part}	t_{input}	$t_{distrib}$	t_{sim}	$t_{collect}$	Σ	Besch.
s1196	C	4	1	2,03	0,26	1,92	0	5,21	0,39
s13207	C	4	18	15,99	4,10	69,31	0,90	108,31	0,64
s35932	C	16	74	49,72	14,81	59,40	0	197,93	1,16
inv64_chain	D	8	1	4,34	0,02	5,95	0	11,31	0,10
inv10000_chain	D	24	40	11,81	2,71	37,34	0	91,86	1,43
inv100000_chain	D	8	3904	458,46	27,08	271,91	0	4661,46	0,17

Tabelle 8.19: Gesamtzeitbedarf der Simulation mit konservativem Verfahren

Diese Ergebnisse relativieren sich wiederum für Applikationsklassen mit erhöhter Granularität. Dort nimmt die Bedeutung der zusätzlichen Kosten mit steigender Ereigniskomplexität stark ab. Für große Modelle wie inv100000_chain können aber insbesondere die Partitionierungskosten nicht vollständig vernachlässigt werden.

Modell	Simulationsdauer								
	Part.	Anz.	t_{part}	t_{input}	$t_{distrib}$	t_{sim}	$t_{collect}$	Σ	Besch.
s1196	C	8	1	2,56	0,29	3,75	0	7,60	0,27
s13207	C	16	17	17,40	5,26	13,90	0	53,56	1,30
s35932	C	16	74	49,72	14,81	90,69	0	229,22	1
inv64_chain	D	8	1	4,34	0,02	5,17	0	10,54	0,11
inv10000_chain	D	16	40	11,77	2,49	41,47	0	95,74	1,37
inv100000_chain	D	8	3904	458,46	27,08	393,72	0	4783,26	0,17

Tabelle 8.20: Gesamtzeitbedarf der Simulation mit optimistischem Verfahren

Neben den offensichtlichen Kosten, die in klar definierten Phasen anfallen, existiert auch eine Klasse von im parallelen Synchronisationsalgorithmus versteckten Overheadgeneratoren. Bei den konservativen Verfahren gehören dazu die Blockadekosten aufgrund mangelnder Garantien und der Aufwand zur Entdeckung von Deadlocks. Insbesondere eine globale Würdigung dieser Kosten ist, wie bereits beim Oracle-log-Verfahren diskutiert, aufgrund der nicht vorhandenen konsistenten Sicht auf das Gesamtsystem problematisch.

Bei den optimistischen Verfahren schlagen überwiegend die Kosten für Rollbacks und wiederholte Ereignisausführung als Overhead zu Buche. Für DVSIM wurde aufgrund der niedrigen Ereignisgranularität weder ein Copy-state-saving, bei dem der komplette Zustand nach jeder Ereignisausführung gesichert wird, noch ein davon abgeleitetes periodisches Sicherungsverfahren (PSS) verwendet. Statt dessen kommt eine inkrementelle Sicherung (ISS) der Änderungen, die von einem einzelnen Ereignis bewirkt werden, zum Einsatz [BSK91a].

ISS hat Vorteile gegenüber PSS, wenn nur kurze Rollbacks auftreten, kleine Teile des Zustands gesichert werden (geringer Sicherungsaufwand) oder viele Rollbacks auftreten [PAW93a]. Diese Beobachtung wird auch in den Arbeiten von Rönngren u.a. sowie West und Panesar experimentell bestätigt. Bei untersuchten Modellen mit weniger als 15 Prozent [RLA96a] bzw. lediglich 2 Prozent [WEP96a] sicherungsrelevatem Anteil schneiden die ISS-Verfahren am besten ab und laufen bis zu

neunmal schneller als die Copy-state-saving-Methoden. [UCC93a] empfehlen diese Vorgehensweise ebenso, falls der Anteil der geänderten Variablen am Gesamtzustand 20 Prozent nicht überschreitet.

In DVSIM entsteht aufgrund der Speicherung des Zustands in den bereits abgearbeiteten Ereignissen und wegen der geringen Granularität kein großer Aufwand für die Zustandssicherung. Bei Verwendung von VHDL-Prozessen kann der Aufwand je nach deren Komplexität allerdings beliebig hoch ausfallen. Zur Speicherung des Zustands von VHDL-Prozessen schlagen Bauer und Sporrer Phantomereignisse vor, die einfach in die Ereignisliste mit eingekettet werden und bei einer Ereignisausführung die Zustandsänderungen protokollieren [BAS93a]. Dieser Ansatz wurde auch in DVSIM vorgesehen aber nicht vollständig für VHDL-Prozesse realisiert.

8.5 Optimierung der Kommunikationskosten unter PVM

Bei der vorliegenden Implementierung wurden die Standardmechanismen von PVM eingesetzt, um die Simulation in beliebigen heterogenen Umgebungen, die diese Architektur unterstützen, ablaufen lassen zu können. Insbesondere wurde dazu die Transformation aller Nachrichteninhalte in das Netzwerkdatenformat XDR [SRI95a] vorgenommen. Dadurch entsteht zusätzlicher Aufwand für das Kopieren zwischen von Sendern bzw. Empfängern benutzten Applikationsdatenpuffern und Puffern der Kommunikationsschicht. Der Aufwand für die eigentliche Konvertierung entsprechend XDR ist dagegen vernachlässigbar, da die Nachrichten aus elementaren Datentypen bestehen.

Für die durchgeführten Untersuchungen wurde eine Optimierung durch Deaktivieren der Konvertierung nicht eingesetzt, da dadurch die Einsetzbarkeit unter allgemeinen Bedingungen auf einer beliebigen PVM-Plattform nicht mehr möglich ist. Bei der Kombination von beispielsweise SUN-Workstations mit PCs innerhalb einer virtuellen Maschine ist bereits eine Datenkonvertierung notwendig.

Alternativ gestattet PVM auch eine direkte TCP/IP-Verbindung zwischen zwei Applikationsprozessen sofern die Betriebssystemressourcen dies erlauben. Durch Verwendung der Option PvmRouteDirect wird die Verteilung der Nachrichten über die rechnerzentralen Instanzen der PVM-Daemonen umgegangen und vermutlich beschleunigt. Diese Option ist auf dem GC/PP implizit durch Nutzung der zugrundeliegenden PARIX-Kommunikationsprimitive [PAR94a] defaultmäßig aktiviert und auf Workstations aufgrund der limitierten Anzahl von Filedeskriptoren, die für TCP/IP-Verbindungen bereitgestellt werden, abgeschaltet.

PVM wird ebenfalls von Kormicki u.a [KMC97a] für synchrone parallele Logiksimulation eingesetzt. Durch Optimierungen wie das Packen von Nachrichten am Ende jedes Simulationsschritts, funktionalen Parallelismus[21] und Unterscheidung der lokalen Ereignisauswertung von der Ereignisbearbeitung externer Events getrennt nach ihren Sendern können bei hoher Aktivitätsrate innerhalb eines Schaltkreises Beschleunigungen von 5,3 auf 5 Workstations erreicht werden. Diese superlinearen Speedups werden nach Aussage der Autoren Cacheeffekten zugeschrieben.

8.6 Auswertungs- und Analysehilfen für parallele Algorithmen

In einer frühen Entwicklungsphase der DVSIM-Umgebung wurde eine Monitoringkomponente in die verteilten Simulatoren integriert, die für eine Protokollierung der ausgeführten Simulationsaktionen (Ereignisausführung, Nachrichtenversand und -empfang, etc.) sorgt, um das Verhalten der Algorithmen zu beobachten, die Leistungsfähigkeit zu beurteilen und Engpässe zu erkennen.

[21] Pipelining der einzelnen Schritte des Algorithmus in der Berechnungsphase wie Ereignislistenoperationen und Ereignisbearbeitung.

Die generierten Daten wurden im Anschluß an die Simulation aus Tracefiles geladen, aufbereitet und mit graphischen Oberflächen visualisiert. Allerdings traten dabei massive Problem mit der Handhabung der Daten auf. Die eingesetzten Visualisierungstools stellten aufgrund der heterogenen Struktur der asynchronen Algorithmen nur moderate Möglichkeiten zur Analyse bereit. Die Kommunikation folgte keinem einheitlichen Muster, das für das Auge des Benutzers klar erkennbar wäre. Lediglich offensichtliche Engpässe wie Phasen sequentieller Abarbeitung auf einzelnen Prozessoren über längere Zeiträume sind erkennbar. Allerdings ist es schwer, eine Korrelation zwischen den gemachten Beobachtungen und der Ursachen alleine aufgrund der graphischen Anzeigen herzustellen. Weiterhin treten Probleme bei der Skalierbarkeit für die meisten Anzeigeverfahren auf. Ein Graph eines Algorithmus mit 16 Prozessoren benötigt zur Anzeige bereits 16 Pixel für ein und denselben Wert, sofern man nicht mit Techniken zur selektiven Anzeige oder Überlagerung arbeitet. Außerdem erfordert die Offline-Bearbeitung die Speicherung von immensen Datenmengen. Die ISCAS-Benchmarks generieren bei ihren kurzen Simulationsläufen bereits 7-stellige Ereigniszahlen. Neben den eigentlichen Simulationsereignissen sind aber auch noch andere Vorkommnisse wie Nullnachrichten, Deadlockerkennung, Rollbacks und Checkpointing von Relevanz, so daß die Datenmenge bei einer minimalen Anzahl von 20 Byte pro Protokolleintrag leicht Hunderte von Megabyte für einen Simulationslauf erreicht. Die Aufbereitung solch großer Datenmengen bereitet auch bei Einsatz intelligenter Filtermechanismen einigen Aufwand.

Hinzu kommt noch die Notwendigkeit der zeitlichen Synchronisation der Beobachtungen, da in der Regel keine überall vorhandenen, gekoppelten Hardwareuhren existieren. Es ist somit wenigstens zu Simulationsbeginn und -ende eine Abstimmung der Uhren für die Protokollierung zu leisten und im Falle nichtlinearer Drift (z.B. auf Sun-Rechnern) können massive Verfälschungen der Beobachtungen auftreten, die die Aussagen über das Verhalten des betrachteten Systems im besten Fall verfälschen. In extren Fällen führen die Fehler zum Programmabsturz von Standardvisualisierungsprogrammen wie ParaGraph [HEF91a], da diese z.B. in der Zeit rückwärts laufende Nachrichten nicht korrekt verarbeiten können.

Eine Online-Visualisierung direkt zur Laufzeit des Simulationsprogramms generiert in der Regel auch keine befriedigenden Ergebnisse aufgrund der nichtdeterministischen Laufzeiten der Monitoringinformationen, die ja ebenfalls über Nachrichten zum Beobachter verschickt werden. Somit ergeben sich bei naiver Herangehensweise für jeden Programmlauf unterschiedliche Beobachtungen, die die Problemanalyse eher erschweren als eine echte Hilfestellung zu bieten.

Ansätze wie das Hardwaremonitoring und die Verwendung von zusätzlichen über eine zentrale Uhr getakteten Hardwareuhren auf allen Prozessoren lagen außerhalb unserer Untersuchungen.

8.7 Beurteilung der parallelen Verfahren

Die Entwicklung des prototypischen VHDL-Simulators DVSIM zeigt ebenso wie früher durchgeführte Projekte [MMR93a, RIC95a], daß Basisverfahren oft erst nach sorgfältiger Optimierung einigermaßen gute Ergebnisse liefern. Reiher [REI92a] spricht bei einer Beurteilung des TWOS davon, daß eine Beschleunigung von 30 auf 72 Prozessoren ein (bis 1992) außergewöhnlich guter Wert ist, der bei irregulären Problemen erreicht wurde, und von anderen Simulatoren bei weitem nicht erzielt werden konnte. Diese Beobachtungen wurden auch hinsichtlich vieler anderer Untersuchungen gemacht. Ein linearer, um 50 Prozent herabgesetzter Speedup ist außerordentlich gut. Die meisten Untersuchungen werten bereits 25 Prozent Effizienz als Erfolg. Allerdings konnten auch sehr widersprüchliche Beobachtungen verschiedener Forscher erklärt werden. Briner, Ellis und Kedems Speedup von 25 auf 32 Prozessoren [BEK91a] beruht vermutlich auf einer hohen Ereignisgranularität während Reed, Malony und McCredies Ergebnisse [RMM88a] unter einer schlechten Ereignismodellierung leiden.

Tuning einer Applikation und die korrekte Wahl vieler Randbedingungen sind somit sehr wichtig. Leider widerspricht das den von Fujimoto gestellten Forderungen nach einfacher Benutzbarkeit paralleler Simulationstools [FUJ93a]. Ein Entwickler erwartet, daß er sich auf die Modellierung konzentrieren kann und das benutzte Tool einfach funktioniert. Eine komplexe Optimierungsaufgabe mit Modifikationen eines ihm ansonsten fremden Systems entspricht nicht seinen Vorstellungen.

Das gleiche gilt auch für sorgfältiges Design, um Determinismus und Korrektheit der Simulation zu erreichen. [RWH92a] und [BAG96a] beschreiben typische Designfehler, die Determinismus und Korrektheit gefährden können. Es stellt sich somit wiederum die Frage nach der Zumutbarkeit solch komplexer Verfahren für einen Simulationsmodellierer ohne Erfahrungen im Entwickeln paralleler Applikationen.

Kapitel 9

Optimierung der partitionsabhängigen Faktoren

Nachdem bisher die Optimierungen auf Seiten der Synchronisationsalgorithmen gesucht wurden, stellt sich in diesem Kapitel die Frage, inwieweit Verbesserungen der Leistungsfähigkeit paralleler Simulationsverfahren auch durch Modifikation der Partitionierungsverfahren erreicht werden können.

Mit den betrachteten Untersuchungsmethoden wurde bei den konservativen und optimistischen Basisverfahren leider nur recht geringe und oft gar keine Beschleunigungen beobachtet. Hinzu kommt der insbesondere bei den konservativen Verfahren relativ hohe Aufwand zur Berechnung günstiger Partitionierungen für die parallele Simulation. Bei größeren Modellen steigen diese Kosten extrem an und überschreiten z.T. die eigentliche Simulationsdauer. Für langlaufende Simulationen mag diese Vorgehensweise noch tragbar sein. Gilt es jedoch zunächst eine gute Partitionierung zu finden, so bereitet alleine die Berechnung der Aufteilungen schon einen hohen Aufwand.

Unter Verwendung verschiedener Optimierungen konnten die Speedup-Werte jedoch auch mit den Basisverfahren gesteigert werden. Zum einen erzielten Simulationen, deren Ereignisgranularität in etwa die Höhe des Aufwands für den Versand einer Ereignisnachricht aufweist, nicht unerhebliche Zeitgewinne. Beim Time Warp konnte weiterhin durch Limitierung des Optimismus mittels Fenstertechniken die in der Basisimplementation des Time Warp beobachteten Probleme mit der Speicherknappheit wenigstens teilweise vermieden und Ergebnisse auf der Qualitätsstufe des konservativen Pendants erreicht werden. Dennoch befriedigen beide Feststellungen nicht. Es stellt sich die Frage, ob es keine besseren Lösungen für feingranulare Modelle gibt und ob die Nebenkosten der Simulation nicht reduziert werden können.

Das Verhalten des limitierten Time Warps legt die Vermutung nahe, daß zum Erreichen einer guten Performance in optimistischen Verfahren die zeitlichen Differenzen der lokalen Uhren auf ein gewisses Maß beschränkt bleiben müssen. Diese Bedingung scheint erfüllt zu sein, wenn die Simulatoren gleichmäßig ausgelastet sind. Für die konservativen Methoden wurden keine algorithmischen Verbesserungen gefunden, die sich in ähnlicher Weise in verbesserter Leistung zeigten[1].

Durch die Integration von Applikationswissen oder dynamisch zur Laufzeit der Simulation ermittelte Daten kann man versuchen, die Qualität der Modellaufteilung zu verbessern. Die erste Variante läßt sich nur eingeschränkt einsetzen. Ein Beispiel hierfür ist die hierarchische Partitionierung, die parallel operierende Komponenten verschiedenen Partitionen zuweist. Mit den verwendeten ISCAS-Schaltungen, über die quasi kein Wissen vorhanden ist und die sich deshalb auch

[1] Verfahren mit Nullnachrichten wurden aufgrund der existierenden Zyklen in den Modellen nicht betrachtet.

nicht hierarchisch anordnen lassen, kann dieser Ansatz nicht verfolgt werden. Für die andere Optimierung durch Laufzeitdaten existieren zwei grundsätzliche Vorgehensweisen. Zum einen können diese Informationen für eine Neuberechnung einer statischen Partitionierungen verwendet werden. Zum anderen besteht die Möglichkeit der Umverteilung von Modellkomponenten zur Laufzeit der Simulation ohne eine vorherige komplexe Berechnung der Ausgangspartitionen.

9.1 Pre-Simulation

Ansätze zur Verbesserung der Leistung paralleler Simulation durch Erhöhung der Qualität statischer Partitionen wurden von Manjikian und Loucks [MAL93a], Chamberlain und Henderson [CHH94a] sowie von Wilson und Nicol [WIN96a] vorgeschlagen. Ziel ist bei allen Verfahren stets, durch Erfassung der realen Ereignisaufkommen die Aktivitätsraten der Modelle besser zu beurteilen. Dazu wird die Simulation für einen kurzen Simulationszeitraum gestartet und Statistiken über die Anzahl der bearbeiteten Ereignisse bei den Objekten bzw. über die Anzahl der für andere Objekte eingeplanten Ereignisse erstellt. Mit diesen Werten werden die Knoten und Kanten des Modellgraphen in einem zweiten Schritt markiert und für eine weitere Iteration der statischen Partitionierung verwendet. Dadurch erhofft man sich die a priori nicht vorhandenen Kenntnisse über das dynamische Verhalten der Modelle besser zu berücksichtigen.

Diese semidynamische Partitionierung[2] greift allerdings nur, wenn das Modellverhalten sich nicht im späteren Verlauf der Simulation drastisch ändert. Untersuchungen in [CHH94a] zeigen jedoch für drei kleinere Schaltkreise, daß die Ereignisraten der Schaltungselemente für die Pre-Simulationsphase recht gut mit den über die gesamte Simulationsdauer auftretenden Werten korrelieren. Das bedeutet, daß sich das Verhalten der betrachteten Schaltungen bereits anhand der ersten Simulationsphase, die zwischen 10 und 30 Prozent der Gesamtlaufzeit betrug, sehr gut approximieren läßt.

Allerdings vermeidet dieses Verfahren nicht die zusätzlichen Kosten für die statische Partitionierung. Zur Ermittlung der zusätzlichen Informationen wird vielmehr weiterer Aufwand für die Pre-Simulationsphase und einen erneuten Zyklus zur Partitionierung unter Verwendung der gewonnenen Daten benötigt.

Diese Vorgehensweise wurde mit DVSIM für den Time Warp mit Modell s1196 überprüft. Bei den Partitionierungsverfahren Soccer und Kernighan/Lin konnten die Leistungen der parallelen Simulation zwar weiter gesteigert werden, allerdings erreichten die Werte lediglich die Ergebnisse der besten statischen Partitionierungsverfahren ohne Einsatz von vorgeschalteten Simulationsläufen. Eine echte Alternative stellen diese Verfahren also nicht dar, da man in diesem Fall ohnehin die bereits günstigeren Partitionierungen verwenden würde. Bei fairer Bewertung der Gesamtkosten der parallelen Simulation sinken durch den Mehraufwand der zweifachen (oder auch häufigeren) Repartitionierung die Beschleunigungswerte noch unter die Ergebnisse der Basisverfahren. Die beiden anderen erwähnten Forschergruppen setzen dieselben Prinzipien ein, um die initiale Partitionierung zu optimieren.

[2]Die Dynamik des Verfahrens ist hier nur an bestimmten fest definierten Stellen möglich, die durch den Benutzer gesteuert werden. Es existiert keine zu jedem Zeitpunkt optimierend wirkende Automatik. Deshalb wurde hier die Bezeichnung semidynamisch verwendet.

9.2 Dynamische Lastbalancierung zur Laufzeit der Simulation

Ein Ausweg zur Vermeidung der statischen Partitionierungskosten bietet hier lediglich deren komplette Abschaffung. Die Simulation startet mit einer zufälligen oder einer mit sehr wenig Aufwand berechenbaren Verteilung ohne Berücksichtigung der Qualität. Durch eine dynamische Lastbalancierung zur Laufzeit wird im Idealfall eine optimale und stabile Verteilung automatisch erzeugt. Diese Vorgehensweise kann insbesondere bei optimistischen Verfahren zu unterschiedlichen Verbesserungen führen, da einerseits die initialen Kosten entfallen und andererseits die Verschiebung eines Objekts auf einen anderen Prozessor ggf. einen Rollback auslöst, der einen überoptimistischen Prozeß bremst und die Abweichungen der lokalen Uhren zwischen den logischen Prozessen (LPs) reduziert.

Zur Steuerung des dynamischen Verfahrens sind vier Größen zu identifizieren: die Definition des Lastbegriffs, die Festlegung eines Schwellwerts, bei dessen Überschreiten eine Rebalancierung erforderlich wird, die Granularität der Umverteilung und die Aktualisierungsrate der Lastinformationen bzw. die Mindestabstände, in denen eine Balancierung vorgenommen werden darf.

Insbesondere die Definition der Last ist keine triviale Aufgabe. Es gibt verschiedene Vorschläge, die sowohl Vor- als auch Nachteile besitzen. Die wichtigsten in der parallelen Simulation verwendbaren Größen sollen hier kurz beschrieben werden.

- Die Prozessorlast bzw. die verbrauchte CPU-Zeit drückt den Aufwand für die Berechnung der Simulationsergebnisse in Bezug auf die benötigten Rechnerressourcen aus. Dabei ist allerdings auch der Overhead der parallelen Verfahren implizit in den Lastwerten enthalten. Ein LP, der unter Time Warp ständig Rollbacks ausführt, ist demnach ebenso belastet, wie ein Prozessor, der Ereignisse bearbeitet, die nicht zurückgesetzt werden müssen.

  Carothers und Fujimoto [CAF96a] beschreiben ein System zur Ausführung von Time-Warp-Simulationen als Hintergrundprozesse (Background Execution), das die Berechnungen in einem nicht exklusiv genutzten Netz von Workstations auf der Basis einer Schätzung der verfügbaren Rechenzeit neu verteilt. Einheit der Migration sind Cluster von LPs. Neue Prozessoren können zum Pool der verfügbaren Rechner dynamisch hinzugenommen werden, wenn sie eine entsprechend niedrige Last haben. Sie ersetzen Prozessoren, die aufgrund von Aktivitäten anderer (z.B. interaktiver) Benutzer stärker belastet sind[3]. Eine statische initiale Partitionierung ist dennoch nötig. Die Cluster selbst ändern sich nach der Festlegung nicht mehr.

- Um ein Maß für den direkten Beitrag einer Berechnung zur Problemlösung zu liefern, definieren Reiher und Jefferson die *effektive Prozessorauslastung*, die für den Time Warp nur den Berechnungsaufwand für Ereignisse beinhaltet, die nicht zurückgesetzt werden und somit zu einem echten Fortschritt der Simulation führen [REJ90a]. Ein niedriger Lastwert bedeutet bei dieser Definition also eine schlechte Auslastung des entsprechenden logischen Prozesses. Die Aufteilung der Prozessorlast erfolgt hier nach dem Space-time-Prinzip (s. Kap. 2.6.1), indem man den Simulationszeitraum, den der belastete logische Prozeß zu simulieren hat, in zwei disjunkte Zeiträume zerlegt. Dabei entsteht ein neuer LP, der auf einen anderen Prozessor verlagert werden kann.

- Die Anzahl simulierter Ereignisse und die Simulationszeit eignet sich bei konservativen Verfahren recht gut als Lastmaß. Bei optimistischen Simulationsmethoden dürfen bei den Ereig-

[3] Allgemeinere Systeme zur Nutzung brachliegender Rechnerressourcen in einem Workstationcluster stellen z.B. CONDOR [LLM87a] und LIPS [SER92a] dar.

nissen nur die sicheren Ereignisse gezählt werden, die aufgrund der fortgeschrittenen GVT nicht mehr zurückgesetzt werden (committed events) [AVT96a]. Die reine Simulationszeit ist für optimistische Methoden dagegen ungeeignet, da sie durch während der Simulation auftretende Rollbacks auch zurückspringen kann.

Burdorf und Marti schlagen deshalb die Bildung eines Mittelwerts der lokalen Simulationszeiten (LVT) für den Time Warp vor [BUM93a]. Luksch grenzt diese Definition noch weiter ein und verwendet die Differenz zwischen mittlerer LVT und der unteren Schranke der global approximierten Simulationszeit (GVT) [LUK93a]. Schlagenhaft u.a. [SRS95a] kalibrieren den Fortschritt der LVT innerhalb konstanter Realzeitintervalle und definieren ihre Last darüber.

Nach erfolgter Definition des Lastbegriffs stellt sich die Frage, wer die Lastbalancierung anstößt. Shivaratri, Krueger und Singhal unterscheiden nach drei verschiedenen Modellen. Eine detailiertere Beschreibung dieser Klassifikation findet sich in [SKS92a]. Hier sollen die Verfahren nur kurz genannt werden. Es handelt sich dabei um die senderinitiierte Balancierung, die empfängergesteuerte Verlagerung und die Kombination dieser beiden Verfahren. Im ersten Fall stellt der Last abgebende Prozeß eine eigene Überlast fest und wählt einen weniger belasteten Partner aus, dem er einen Teil seiner Berechnungen übergibt. Die zweite Varianten erkennt, daß lokal nicht genügend Arbeit vorhanden ist und fragt bei einem oder mehreren anderen Prozessen nach, ob diese nicht einen Teil ihrer Arbeit abgeben können.

Für optimistische Verfahren gibt es bislang wenige Ansätze zur dynamischen Lastbalancierung, die jedoch meist auf der Balancierung fest definierter Cluster basierten. Dazu wurde vor der eigentlichen Simulation zusätzlich wie bei der statischen Partitionierung eine Aufteilung der Schaltung berechnet, die jedoch aus wesentlich mehr Teilen als vorhandenen Prozessoren besteht. Ein LP simuliert zur Laufzeit mehrere Cluster. Bei Lastungleichgewichten werden komplette Cluster verlagert [SRS95a, AVT96a]. Dadurch erspart man sich zwar die dynamische Berechnung zu verlagernder Elemente zur Laufzeit, aber es fallen wiederum die Kosten für die initiale statische Partitionierung an. Außerdem kann nicht flexibel auf dynamische Verhaltensänderungen der Schaltungen zur Laufzeit reagiert werden[4]. Zum Teil verwenden diese Lösungen auch einen zentralen Ansatz zur Auswahl der an der Umverteilung beteiligten Partner, was zu einem Engpaß bei der Lastermittlung, die durch Nachrichtenaustausch mit einer zentralen Komponente erfolgt, führen kann.

DVSIM verwendet dagegen einen vollständig dynamischen Ansatz, der ohne zentrale Kontrolle und ohne kostspielige initiale Partitionierung auskommt. Diese Vorgehensweise liegt nahe, da auf jedem Prozessor nur ein einziger LP vorhanden ist, der eine Vielzahl von Gattern simuliert. Unter diesen Annahmen soll in diesem Kapitel untersucht werden, ob zusätzliches Beschleunigungspotential existiert. Es wird zunächst der Grundalgorithmus zur Lastbalancierung beschrieben und seine Vorgehensweise für die optimistischen Verfahren diskutiert. Für konservative Methoden ist bislang nur eine Beschreibung eines Algorithmus zur dynamischen Lastbalancierung bekannt, der ebenfalls auf der Migration kompletter Cluster basiert [BOD97a]. In einer Fortführung unseres Verfahrens für die optimistischen Ansätze wird der Algorithmus auf die konservativen Verfahren ausgeweitet und Argumente für die Funktionsfähigkeit geliefert.

9.3 Lastbalancierung mit DVSIM für optimistische Simulation

Die Lastbalancierungskomponente von DVSIM wird durch eine hohe Anzahl von Freiheitsgraden bestimmt, die möglichst viele Parameter bei der Lastumverteilung nicht vorzeitig belegen. Zum

[4]Die Gültigkeit der Pre-Simulationsexperimente in [CHH94a] für beliebige Modelle wird bei dieser Argumentation nicht vorausgesetzt.

einen existiert keine zentrale Kontrolle, die sich im Nachhinein als Engpaß erweisen könnte. Daneben läßt sich über Parametrisierung der Simulationsläufe die Schwelle zur Festlegung der Lastdifferenz und die Anzahl der zu verschiebenden Gatter definieren, sowie die Häufigkeit der Umverteilung steuern.

Als einzige Reglementierung wurde eine Senderinitiierung des Algorithmus verwendet (d.h. ein LP stellt seine Überlast fest und reagiert darauf) und eine Limitierung der gleichzeitigen Lastaustauschvorgänge auf maximal eine Verschiebung vorgenommen. Die Fairneß bei der Lastverteilung wird durch ein Token gewährleistet, das sukzessive alle LPs entlang eines virtuellen Rings besucht. Durch diese Restriktionen soll vermieden werden, daß ein unterbelasteter LP durch gleichzeitigen Versand von Modellteilen durch mehrere andere LPs schlagartig in einen Überlastbereich gerät und auch alle LPs eine Chance bekommen, bei Bedarf Last abzugeben.

Zur Definition des Lastmaßes im Time Warp wurde das Modell von Luksch [LUK93a] verwendet. Die mittlere LVT wird stets über mehrere Meßpunkte berechnet. Die Daten werden dabei unmittelbar bei Eintreten eines Rollbacks und vor dem Zurücksetzen der LVT aufgezeichnet. Sie entsprechen somit dem maximalen Fortschritt der Simulation zwischen zwei benachbarten Rollbacks. Die Last wird dann als Differenz aus der aktuellen GVT und diesem Mittelwert berechnet.

9.3.1 Aufbau des Lastbalancierungsmoduls

Um später eine weitgehend problemlose Integration und homogene Struktur des Lastbalancierungsverfahrens auch für die konservativen Verfahren zu gestatten, wurden bereits in der optimistischen Variante einige Designentscheidungen getroffen, die für beide Algorithmenklassen relevant sind. In den folgenden Beschreibungen wird an geeigneter Stelle bereits auf Parallelen beziehungsweise Unterschiede hingewiesen.

Der LP, der zur Balancierung der Last berechtigt ist, fordert die Lastwerte seiner Nachbarn an. Daraus berechnet er die Differenz zur eigenen Last und wählt bei Überschreiten der Toleranzschwelle (z.B. 10 Prozent Unterschied) denjenigen LP mit dem geringsten berichteten Lastwert aus. Nachdem ein LP den entsprechenden Partner für die Migration ermittelt hat, wird anhand des Parameters, der die Anzahl der zu verschiebenden Gatter festlegt, eine Teilpartition bestimmt, die an den weniger belasteten LP verschickt werden soll.

Die Auswahl erfolgt dabei so, daß stets solche Gatter am Rand der Partition ausgewählt werden, die mindestens ein Eingangs- oder Ausgangssignal bereits in der Zielpartition besitzen. Dadurch reduziert sich das Risiko, daß neue Zyklen zwischen den beteiligten LPs eingeführt werden, die dann zu einem erhöhten Nachrichtenaufkommen führen könnten. Eine Kontrolle der globalen Zyklenfreiheit der neu entstehenden Aufteilung wird nicht durchgeführt. Aufgrund der Clusterung aller logischen Kanäle (Signale zwischen zwei Objekten, die auf unterschiedlichen LPs plaziert werden) innerhalb eines einzigen virtuellen Kanals für die beiden betroffenen LPs, ensteht somit in Abb. 9.1 durch die Verschiebung von Gatter b von LP_1 auf LP_2 ein vorher nicht vorhandener Zyklus, der sich bei einem konservativen Verfahren zudem negativ in vermehrten Deadlocks auswirken kann.

Stehen die zu verschiebenden Objekte fest, so müssen diejenigen Simulatoren, die die betroffenen Gatter treiben, über die Verschiebung benachrichtigt und die Routinginformation aktualisiert werden. Danach erhalten die neuen Besitzer ebenfalls die zukünftigen Ereignisse und die korrekte Behandlung der verteilte Knoten (siehe Abb. 4.6) ist auch nach der Migration sichergestellt. Sind alle Signale auf dem Ziel bereits vorhanden, weil sie schon von anderen Gattern benötigt werden, so erübrigt sich dieser Schritt. Die treibenden Gatter von zu verschiebenden Signalen werden zusätzlich vor der Migration als selbst nicht verschiebbar markiert, damit die Aktualisierung der Routinginformation nicht zu aufwendig wird.

Die zu verlagernden Gatter nehmen die Informationen über die bereits ausgeführten und noch auszuführenden Ereignisse ihrer Eingangs- und Ausgangssignale mit auf den neuen LP. Ist ein Signal noch nicht vorhanden, so wird diese Information aus den mitgebrachten Daten rekonstruiert. Andernfalls können die redundanten Daten aus der Migrationsnachricht einfach gelöscht werden. Alternative Ansätze wie das Anfordern der Zustände vom ursprünglichen LP im Falle eines Rollbacks erfordern einen zu hohen Verwaltungsaufwand, da stets gespeichert werden muß, auf welchem Simulator sich ein Objekt in welchem Zeitraum aufhielt. Darüberhinaus verwendet DVSIM inkrementelle Zustandssicherung innerhalb der abgearbeiteten Ereignisse, so daß der Mehraufwand für die Übertragung dieser Information bei der Verschiebung nicht sehr groß ist.

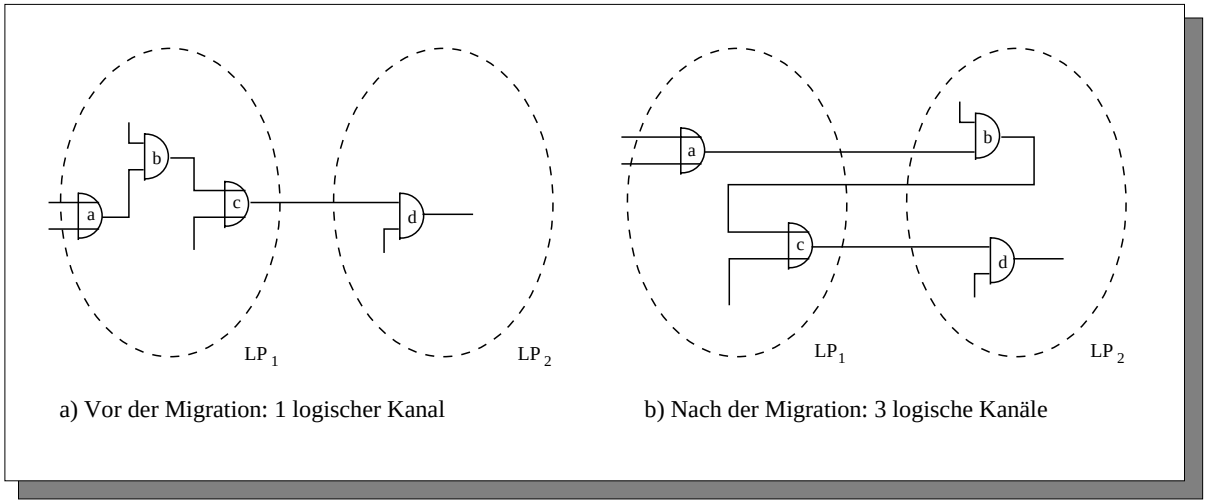


Abbildung 9.1: Entstehung von Zyklen durch fehlerhafte Gatterauswahl

Ein besonderes Problem stellen die Nachrichten dar, die im Zeitraum zwischen dem Versand der Gatter und ihrer Reaktivierung auf dem Ziel-LP noch zum alten Besitzer unterwegs sind. Diese Nachrichten müssen zum neuen Ziel weitergeleitet werden, da die Existenz des zugehörigen Signals auf dem neuen Ziel nicht vorausgesetzt werden kann. Der ursprüngliche Besitzer der Gatter verwaltet dazu die Informationen über das neue Ziel und reicht verspätet ankommende Nachrichten unbehandelt weiter. Um die Administration dieser Informationen auch nach mehreren Migrationen noch wartbar zu halten, muß jeder LP, der über die Verschiebung eines von ihm getriebenen Gatters informiert wird, für den betroffenen Knoten eine Flush-Nachricht auf den unter PVM verwendeten FIFO-Kanälen verschicken. Anschließend wird nur noch das neue Ziel bedient. Nach Eintreffen einer Flush-Nachricht kann der ursprüngliche Besitzer den Vermerk für die Weiterleitung löschen, da er sicher sein kann, daß für das verschobene Gatter nun keine weiteren Nachrichten mehr eintreffen. Der Flush-Marker selbst wird auch noch an das neue Ziel weitergereicht, damit der Ziel-LP die Freigabe der blockierten Treiberobjekte initiieren kann.

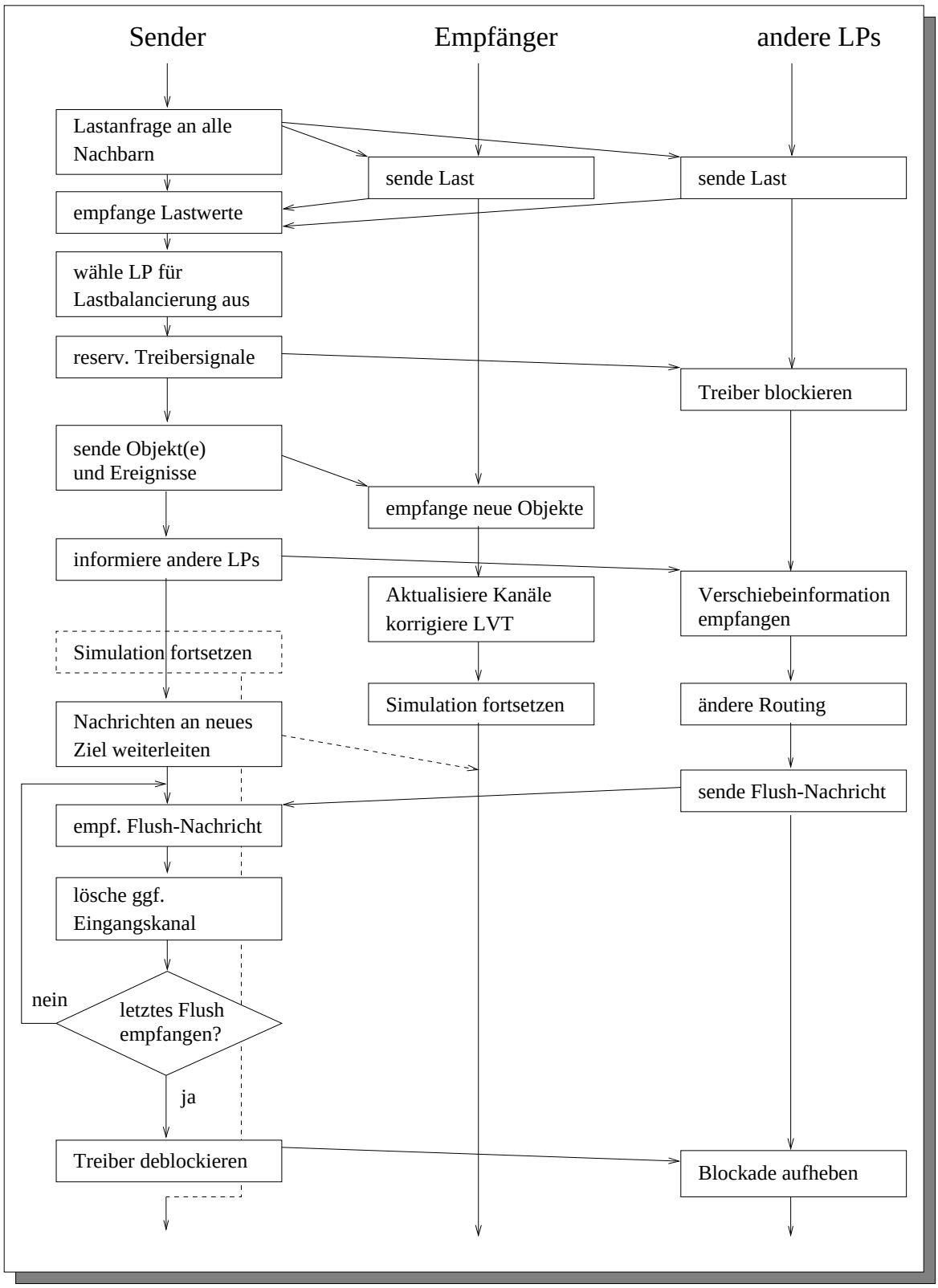


Abbildung 9.2: Ablauf der Migration

Die Alternative, zunächst auf das Eintreffen der Flush-Nachrichten zu warten und erst anschließend die Migration vorzunehmen, führt zu einem anderen Problem. Da der Treiber bereits über die Migration informiert ist, sendet er nun alle Ereignisnachrichten an die neue Lokation. Diese kennt allerdings die betroffenen Signale ggf. noch gar nicht, so daß die neuen Ereignisse hier gepuffert werden müssen, bis das Gatter wirklich eintrifft[5]. Das Migrationsprotokoll läuft also insgesamt in drei Phasen ab:

1. Blockierung der treibenden Gatter für alle von der Migration betroffenen Signale,
2. (a) Verschieben der Gatter zum neuen Ziel,
   (b) Information der anderen LPs über die Topologieänderung und
   (c) Weiterleitung von Nachrichten.
3. Nach Empfang der ausstehenden Flush-Nachrichten:
   (a) Beenden der Weiterleitung und
   (b) Freigabe der treibenden Gatter.

Die einzelnen Aktionen in den verschiedenen Phasen laufen dabei in der Regel verschränkt zueinander und transparent für die eigentlichen Simulation ab. Prinzipiell ist auch die gleichzeitige Migration durch mehrere LPs denkbar. Dann müssen jedoch die Treibersignale mehrfach blockierbar sein und eine Kollisionserkennung für Signale, die von mehreren Migrationswünschen betroffen sind, ist nötig. Wegen der Übersichtlichkeit der Ergebnisanalyse wurde die Dynamik aber wie bereits erwähnt auf eine gleichzeitige Verschiebung beschränkt.

Die durchzuführenden Aktionen auf den betroffenen LP-Typen "Sender" und "Empfänger der neuen Gatter" und "andere LPs" sind in Abbildung 9.2 dargestellt. Die Aktualisierung der Kanäle wird nur für das konservative Verfahren benötigt (siehe Kap. 9.5). Die Korrektur der LVT erfolgt im Time Warp bei Bedarf über einen Rollback.

9.3.2 Ergebnisse

Mit dieser Realisierung der Lastbalancierungskomponente wurden Untersuchungen für die optimistischen Verfahren durchgeführt. Insbesondere interessierte, inwieweit sich zu statischen Partitionierungen vergleichbare Ergebnisse erreichen ließen. Trotz einer breiten Palette unterschiedlicher Parametrisierungen konnten jedoch keine weiteren Verbesserungen der Laufzeiten erzielt werden. Insbesondere erfüllte sich auch die Hoffnung, mit einer billigen Startpartitionierung zu beginnen und daraus weitere Beschleunigungen ohne aufwendige initiale Modellverteilung zu erhalten, leider nicht.

Bei Start der Simulation mit einer im letzten Kapitel verwendeten Modellaufteilungen wurden für die realen Modelle s1196, s13207 und s35932 durch die Lastbalancierungskomponente in den meisten Fällen tatsächlich Lastdifferenzen festgestellt und Gatter ausgetauscht. Diese Beobachtung fällt jedoch weitgehend in die Startphase der Berechnung. Im Anschluß daran blieben die Partitionen weitgehend stabil. Vermutlich hat die initiale Repartitionierung einen ähnlich dämpfenden Einfluß wie die eingesetzten Fenstertechniken. Die lokalen Simulationszeiten bleiben durch

[5]Diese Problematik kann sich allerdings aufgrund der nichtdeterministischen Nachrichtenlaufzeiten in einigen besonders konstruierten Fällen beim zuerst geschilderten Verfahren auch ergeben. Allerdings ist das Überschneidungszeitfenster dort sehr klein.

die Rollbacks in der Startphase dichter beieinander, so daß sich die LPs im weiteren Verlauf durch nachfolgende Rollbacks stärker gegenseitig kontrollieren.

Die Simulationsläufe benötigten in den meisten Fällen aufgrund zusätzlicher Migrationskosten zwischen zwei- und dreimal so lange wie die Basisversion. Das beste Ergebnis lieferte Modell s13207 mit azyklischer Startpartitionierung auf 4 Knoten. Allerdings werden hier auch nur zwei Gatter während einer Simulationsdauer von 68 Sekunden ausgetauscht, was einem zusätzlichen Aufwand von 12 Prozent entspricht. In fast allen anderen Fällen wird ein erheblichen Mehraufwand in der Lastbalancierungsversion generiert. Auch bei Simulation der künstlichen Schaltung inv10000_chain konnten keine Verbesserungen der Beschleunigungszeiten erreicht werden. Bei Start mit einer Round-robin-Partitionierung lagen die Laufzeiten ebenfalls wesentlich über denen der Basisversion mit derselben initialen Aufteilung.

Anhand der Experimente wird deutlich, daß eine weitere Leistungssteigerung mit dynamischer Lastbalancierung unter Verwendung einer vollständig dezentralen Kontrolle unwahrscheinlich ist. Es wurde bei der Entwicklung mit erheblichem Aufwand versucht, die Realisierung dieses Moduls effizient zu gestalten, wobei weitere Optimierungen aber noch denkbar sind. Allerdings scheinen der Aufwand dafür und die nach aktuellem Kenntnisstand eventuell noch möglichen Verbesserungen in einem derart starken Mißverhältnis zu stehen, daß im Rahmen dieser Arbeit keine weiteren Modifikationen des Algorithmus durchgeführt und untersucht wurden.

Nachfolgend findet sich eine Auflistung der verwendeten Parameter. Dabei wurden die wesentlichen Kombinationen in der angegebenen Reihenfolge überprüft, indem für die besten Ergebnisse der aktuellen Meßreihen der nächste Parameter variiert wurde. Die Erhöhung der Granularität wurde ebenfalls kurz angetestet. Es konnten aber auch hiermit keine Verbesserungen erreicht werden.

1. Synchronisationsalgorithmus: Lazy-cancellation, Aggressive-cancellation.
2. Minimaler Abstand zwischen zwei Lastneuberechnungen: 1 ms (quasi ununterbrochen), 10 ms, 100 ms, 1 Sekunde.
3. Anzahl der pro Migration verschobenen Gatter: 2, 10, 50, 100.
4. Rechnerplattform: GC/PP und Workstation-Cluster.

Die einzige merkliche Verbesserung, die festgestellt werden konnte, bestand wiederum darin, daß einige Modelle seltener mit Speichermangel abgebrochen wurden. Um diesen Effekt zu erreichen, ist es aber sinnvoller, die Limitierung des Optimismus über Fenstertechniken zu steuern, da dort auch noch einigermaßen vertretbare Beschleunigungswerte bei einem weitaus geringeren Implementierungsaufwand erzielt werden.

9.4 Arbeiten anderer Gruppen

Einige andere Forschergruppen beschäftigen sich ebenfalls seit einigen Jahren mit dynamischer Lastbalancierung für parallele Simulation. Boukerche und Das geben eine Übersicht in [BOD98a]. Die Projekte unterscheiden sich von DVSIM im wesentlichen in der Tatsache, daß beginnend mit einer initial guten statischen Partitionierung die Leistung weiter verbessert werden soll. Die Wahl der zu verschiebenden Komponenten ist dabei oft auf vordefinierte Cluster beschränkt. Die wenigen bekannten und schwer vergleichbaren Ergebnisse werden hier kurz beschrieben.

Reiher und Jefferson erreichen Verbesserungen des Time Warp zwischen 6 und 19 Prozent auf ihrem Space-time-Simulator [REJ90a]. Die LB-Komponente wird dabei alle zwei Sekunden

angestoßen. Die Dauer einer Migration beträgt bei einem einzigen Objekt 0,5 Sekunden und bei einer Gruppe von 32 Objekten 0,75 Sekunden. Höhere Beschleunigungen wurden bei gleichzeitiger Migration mehrerer Objekte beobachtet.

In [SRS95a] werden Beschleunigungen von 25 Prozent gegenüber der Time Warp-Variante für s13207 bei dynamischer Lastbalancierung mit zwei Prozessoren erreicht. Bei Einsatz von lediglich zwei Prozessoren treten jedoch keine Probleme bei der Koordination mit weiteren LPs auf. Es ist schwer abzuschätzen, wie die Erwartungen bei Verwendung größerer Rechnerkonfigurationen ist. Migriert werden stets vordefinierte Cluster aus 200 bis 400 Gattern, wobei mehrere Cluster innerhalb eines LPs angesiedelt sind. Die Kontrolle der Last erfolgt durch eine zentrale Komponente.

Einen ähnlichen clusterbasierten Ansatz verfolgen auch Avril und Tropper [AVT96a]. Bei ihnen wird jedoch jeder Cluster als separater logischer Prozeß modelliert. Hier werden für die ISCAS'89-Modelle s38417 und s38584 Beschleunigungen von etwa 40 bzw. 25 Prozent mit 20 Prozessoren gegenüber der Basisversion erreicht.

Jones und Das schließlich bildeten ein Time Warp-System mit Hilfe der sequentiellen Simulationsumgebung CSIM nach [JOD98a]. Es wurden idealisierte Annahmen zu Simulations- und Kommunikationskosten getroffen und ein synthetisches Warteschlangenmodell mit 512 LPs simuliert. In Abhängigkeit von den Kosten der Migration wurde eine Kombination aus Fenstertechniken mit fester Größe der Fenster und dynamischer Lastbalancierung entwickelt, die bei hohen Migrationskosten die konventionelle Limitierung des Optimismus einsetzt und andernfalls die Lastbalancierung verwendet. Beschleunigungen von 6 auf 32 Prozessoren werden für sehr restriktive Fenstergrößen von 1/4 des mittleren Ereignisabstands berichtet. Bei billiger Migration werden Beschleunigungen von 14 auf 32 Prozessoren erwartet. Eine Einordnung dieser teilweise theoretischen Ergebnisse fällt schwer.

Insgesamt ergibt sich aus diesen und den eigenen Ergebnissen, daß eine vollständige dezentrale Kontrolle eines Lastbalancierungsalgorithmus zu aufwendig erscheint. Die Gefahr eines Engpasses durch eine zentrale Kontrolleinheit ist nicht so immanent, wie ursprünglich befürchtet. Leider liegt wenig Literatur über obige Projekte vor, aber die berichteten Ergebnisse stellen ein Verbesserungspotential relativ zu den reinen Time Warp-Implementierungen mit den vorgestellten Algorithmen dar. Eine Einordnung im Vergleich zu einem schnellen sequentiellen Simulator wird jedoch in keiner der vorliegenden Arbeiten vorgenommen, so daß hier keine Aussage über das absolute Beschleunigungspotential getroffen werden kann.

9.5 Dynamische Lastbalancierung für konservative Simulation

Bei der Planung der Lastbalancierungskomponente wurde bereits überlegt, inwieweit eine Integration auch in ein konservatives Verfahren Sinn macht. Auch wenn keine Implementierung in DVSIM mehr vorgenommen wurde, da die Komplexität des Algorithmus noch über dem der optimistischen Realisierung liegt, soll hier kurz eine Skizzierung des Verfahrens vorgenommen und eine Begründung für seine Korrektheit gegeben werden.

Prinzipiell arbeitet die konservative Variante analog zu der optimistischen Version. Allerdings wird aufgrund der benötigten Garantien im konservativen Modell die Freigabe der Ereignisse wesentlich restriktiver gehandhabt. Der kritische Fall besteht im konservativen Verfahren darin, daß ein Gatter von einem LP mit hoher Last (d.h. mit niedriger Simulationszeit) zu einem Prozessor mit niedriger Last (d.h. hoher Simulationszeit) verschoben wird. In diesem Szenario trifft das zu verschiebende Gatter in der lokalen Vergangenheit des Zielsimulators ein und könnte kausale Verletzungen der Ereignisreihenfolge bewirken, da es ggf. Ereignisse mitbringt, die kleinere Zeitstempel als die lokale Simulationszeit tragen.

Betrachtet man jedoch das praktische Verhalten einer konservativen Simulation, in der keine zusätzlichen Informationen über den frühsten Zeitpunkt des nächsten ausführbaren Ereignisses vorliegen (Lookahead = 0), so kann man zwei Fälle unterscheiden. In zyklenfreien Modellen besitzt die lokale Uhr eines LPs, der von anderen LPs Ereignisse eingeplant bekommen kann, stets einen kleineren Zeitstempel als der kleinste Wert der Uhren aller ihn treibenden LPs. Andernfalls wäre das konservative Prinzip verletzt, nach dem alle Ereignisse erst dann ausgeführt werden, wenn höhere Garantien an allen Eingangskanälen anliegen.

Das heißt nun wiederum, daß eine Verlagerung, bei der ein Gatter in der lokalen Vergangenheit beim Ziel ankommt, nur in Richtung der Eingangskanäle erfolgen kann. Da dieses Gatter aber von seinem neuen Ziel getrieben wird, sind alle Ereignisse, die für dieses Gatter jemals erzeugt werden, bereits produziert. Es kann somit gar keine späteren Ereignisse geben, die kausale Rückwirkungen auf die Vergangenheit des Ziels der Migration nehmen.

Nach einem einfachen Zurücksetzen der lokalen Simulationszeit beim Ziel-LP auf den Zeitstempel des sendenden LPs und dem Eintreffen der Flush-Nachrichten, kann die Simulation einfach fortgeführt werden. Für die Simulationszeitspanne zwischen dem neuen und dem (höheren) alten Wert werden zunächst lediglich Ereignisse des neu empfangenen Gatters simuliert. Beim nachfolgenden Überschreiten der ursprünglichen Simulationszeit sind dann alle vorproduzierten Ereignisse des verschobenen Gatters abgearbeitet und die Simulation läuft wie gewohnt weiter. Damit die konservative Kontrolle für ein verschobenes Gatter funktioniert, müssen ggf. auch noch Kanäle für die neuen und vor der Migration noch nicht vorhandenen Signale angelegt werden. Die Zeitstempel der Kanäle werden vom Initiator der Migration bei der Gatterübertragung mitgeliefert.

Eine ähnliche Argumentation gilt, wenn man Zyklen im Modell besitzt. Dann sind die Werte der lokalen Uhren aller LPs innerhalb eines Zyklus durch den Wert des zuletzt ausgeführten Ereignisses bestimmt, da kein LP einem anderen eine bessere Garantie als den Wert seiner lokalen Uhr geben kann. Die Ausführung jedes beliebigen Ereignisses erfordert somit die Erkennung und Auflösung eines Deadlocks. Die Verschiebung eines Gatters innerhalb des Zyklus erfolgt deshalb zwischen LPs mit gleichem Zeitstempel, was kein Problem darstellt. Verläßt ein Gatter durch die Migration einen Zyklus, so liegt wieder der erste Fall vor.

Existieren zusätzlich Garantien (mit einem Lookahead > 0), so läßt sich das Verfahren ebenfalls durchführen. Jedoch wird die Aktualisierung der Kanalzeiten dadurch komplexer, da neben den Kanalzeiten des Initiators auch die Garantien der betroffenen Eingangssignale integriert werden müssen. Es stellt sich somit die Frage, inwieweit der dadurch generierte Zusatzaufwand sich noch rentiert und ob damit noch effektive Beschleunigungen erreicht werden können.

Dennoch ist die theoretische Vorstellung eines in einem konservativen Verfahren durchführbaren Rollbacks (das entspricht ja gerade dem Zurückgehen in der Simulationszeit) ohne jedoch eine Restauration des Zustands vornehmen zu müssen, ein sehr interessanter Gedanke, der jedoch nicht in die Praxis umgesetzt und genauer untersucht wurde.

Kapitel 10

Skalierbarkeit paralleler Simulation

Eine wichtige Problemstellung zu Beginn der Untersuchungen des Beschleunigungspotentials bei der Parallelisierung ereignisgesteuerter Simulation war auch die Frage, wie gut diese Anwendungsklasse skaliert, d.h. ob die Ergebnisse auch bei wachsenden Problemgrößen stabil bleiben. In unserem Fall drückt sich die Problemgröße in der Anzahl der in einem Modell vorhandenen Schaltungselemente aus. Bezüglich der realen Modelle wurden angesichts der aktuellen Spitzenwerte der Entwurfsgrößen von einigen Millionen Transistoren pro Prozessor eher kleine Modelle untersucht, die gerade ein Hundertstel dieses Wertes erreichen. Allerdings wird in realen Entwurfsprozessen die Simulation zum Teil auch hierarchisch durchgeführt, indem die logisch zusammenhängenden Schaltungsteile separate überprüft und anschließend auf einer höheren Ebene Tests des korrekten Zusammenspiels der Komponenten durchgeführt werden. Dennoch erreichen die Modellgrößen auch hier noch Dimensionen von einigen hunderttausend Gattern, wenn es darum geht, Effekte wie unerwünschte Hazards auszuschließen. Bei homogeneren Schaltungsentwürfen, z.B. bei Speicherchips, ist es unter Umständen nur nötig, einen Ausschnitt der (identischen) Speicherzellen zusammen mit der Ansteuerungslogik zu simulieren. Dadurch wird die Komplexität der Simulation ebenfalls reduziert.

Ein zweiter Skalierbarkeitsaspekt bei der parallelen Simulation betrifft das unterliegende Kommunikationssystem. Insbesondere auf einem busbasierten Netzwerk könnte die Erhöhung der Prozessoranzahl und der Modellkomplexität zu starken Leistungseinbußen führen.

In diesem Kapitel soll es um die Beantwortung dieser beiden Fragestellungen gehen. Zunächst wird anhand großer synthetischer Schaltungen überprüft, ob die mit den mittleren Modellen erreichten Beschleunigungen auch mit großen Schaltungen möglich sind. Anschließend werden die auf dem GC/PP durchgeführten Simulationen mit den realen Modellen auf ein lokales LAN (Ethernet mit 10 Mbit/s) unter Verwendung von SUN SparcStations[1] übertragen. Die verwendeten Sparc-Prozessoren besitzen bezüglich ihrer Leistungsfähigkeit in etwa dieselben Charakteristika wie die PPC 601-Knoten des GC/PP[2]. Da nur eine beschränkte Rechneranzahl zur Verfügung stand, wurden dabei überwiegend Messungen mit 4 und 8 Knoten durchgeführt.

10.1 Skalierbarkeit der Berechnungen

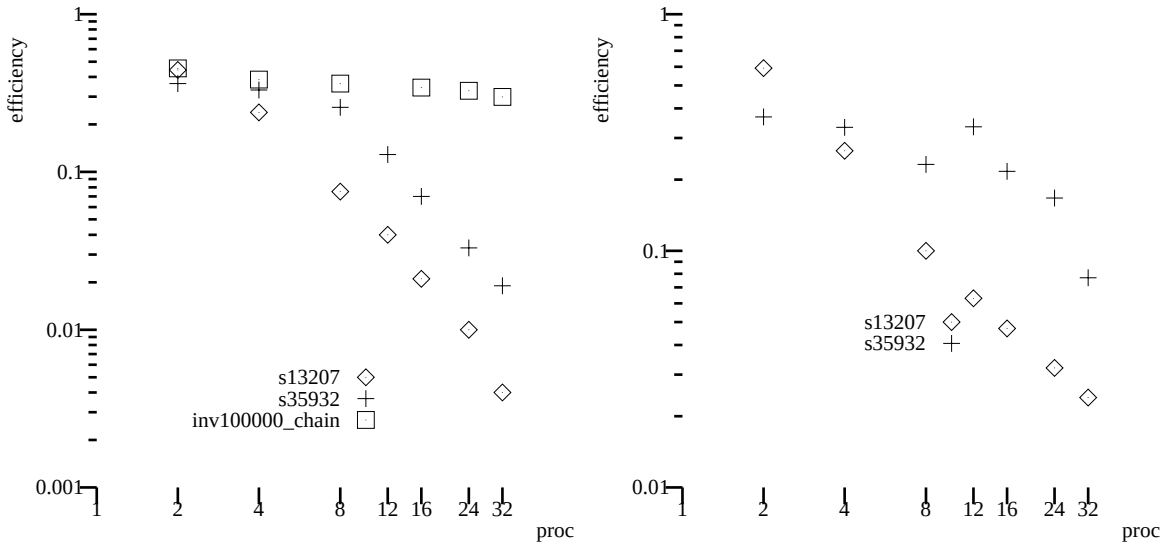
Kumar definiert als Bewertungsmodell für die Skalierbarkeit datenparalleler Algorithmen die sogenannte Isoeffizienz [KGR94a]. Sie gibt an, in welcher Größenordnung die Problemgröße relativ zum linearen Wachstum der Prozessoranzahl steigen muß, so daß die Effizienz (Verhältnis von Speedup

[1] Eine Doppelprozessor SparcStation20, 2 SparcStation 10 und mehrere SparcStation 4.
[2] Die Laufzeiten von VSIM stimmen auf beiden Plattformen nahezu überein.

zur Prozessoranzahl) der verteilten Berechnung konstant bleibt. Erreicht man also bei der Verdoppelung der Problemgröße mit der doppelten Prozessoranzahl dieselbe Effizienz der Lösung, so liegt lineare Isoeffizienz vor. Ist eine Vervierfachung oder Verachtfachung der Aufgabengröße nötig, so erhält man quadratische bzw. kubische Isoeffizienz. Dieses Maß geht davon aus, daß der Zusatzaufwand der verteilten Berechnung sich in der Effizienz widerspiegelt. Isoeffizienzen höheren Grades deuten auf ineffiziente oder aufwendige Koordinationsmaßnahmen bei der verteilten Abarbeitung hin.

Dieses Maß wurde eigentlich für hochgradig datenparallele Probleme definiert. Bei geeigneter balancierter Partitionierung oder bei grobgranularen Ereignissen kann dieses Maß jedoch auch auf die bei der ereignisgesteuerten Simulation vorherrschenden heterogenen und irregulären Struktur angewandt werden.

Bereits bei Basisgranularität zeigt sich, wenn auch auf sehr niedrigem Effizienzniveau, für die Modelle s13207 und s35932 ein recht gutmütiges Skalierbarkeitsverhalten. Bei einem Größenverhältnis der beiden Modelle von 1 : 2,5 bleibt die Effizienz für größenbalancierende Partitionierungen wie Soccer oder kegelbasierte Verfahren bereits bei Verdoppelung der Prozessoranzahl auf demselben Wert, womit sich eine nahezu lineare Isoeffizienz ergibt (Abb. 10.1a). Bezieht man das nochmals um den Faktor 2,5 größere Modell inv100000_chain ebenfalls mit ein, so ergeben sich noch weitaus bessere Ergebnisse, die allerdings nicht unbedingt auf reale Modelle übertragbar sind[3].



a) balancierte FanOut-Kegel b) azyklisch

Abbildung 10.1: Isoeffizienz der untersuchten Modelle, konservative Simulation

Bei azyklischer Partitionierung der Schaltungen machen sich die ungünstigen Aufteilungen bei s13207 zusätzlich bemerkbar (Abb. 10.1b). Selbst bei einer Vervielfachung der Prozessoren ergeben sich bei s35932 immer noch dieselben Effizienzwerte. Negativ ausgedrückt bedeutet dies, daß das kleinere Modell bei Verdopplung der Prozessoren wesentlich mehr Aufwand generiert, als dies bei s35932 der Fall ist.

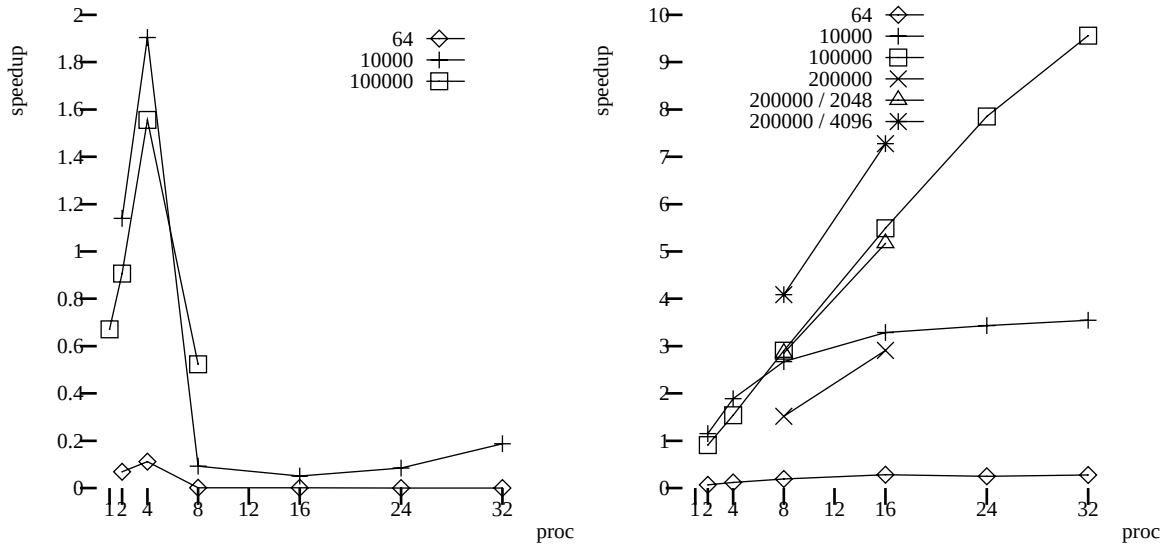
Die Etablierung der allgemeinen These, daß parallele ereignisgesteuerte Simulation generell gut skaliert, ist aufgrund der recht eingeschränkten Untersuchungen nicht möglich. Bailey beobachtete bei Untersuchungen an Schaltkreisfamilien z.T. ein ähnliches Verhalten bezüglich des in den

[3]Leider stand uns kein größeres Modell im Rahmen der ISCAS-Suite zur Verfügung.

Schaltungen vorhandenen Parallelismus (Anzahl gleichzeitig ausführbarer Ereignisse) [BAI92a]. Sie konnte jedoch auch keine eindeutigen Schlüsse ziehen, da bei vier von fünf Schaltungstypen keine klaren Tendenzen beobachtet wurden. Allerdings basieren ihre Untersuchungen auf der Anzahl der Ereignisse mit gleichem Zeitstempeln, während bei unseren Betrachtungen alle zeitgleich in der realen Zeit ausgeführten Ereignisse berücksichtigt werden. Der ermittelte Wert ist zwar empirisch, gibt aber vermutlich auch exakter den vorhandenen Parallelismus wieder, weil auch Ereignisse mit verschiedenen Simulationszeitstempeln einfließen.

10.2 Skalierbarkeit in Abhängigkeit der Modellgröße

Es stellt sich die Frage, wie sich eine Modellfamilie bei steigender Modellgröße verhält. Als Beispielschaltung wurden Vertreter der synthetischen Benchmarks benutzt, die lediglich eine einzige lange Kette darstellen (inv$n$_chain). Abb. 10.2 stellt die Ergebnisse für das konservative Simulationsverfahren auf dem GC/PP für die Soccer- bzw. die balancierte FanOut-Kegelpartitionierung bei verschiedenen Schaltungsgrößen dar.



a) Soccer b) balancierte FanOut-Kegel

Abbildung 10.2: Verhalten bei wachsender Modellgröße, Chain-Modelle, GC/PP, konservativ

Auch hier bestätigen sich die Beobachtungen, daß Beschleunigungen erst ab einer gewissen Größe der einzelnen Partitionen erreicht werden können. Die kleine Schaltung mit 64 Gattern wurde nur der Vollständigkeit halber dargestellt. Die Resultate liegen bei den einzelnen Partitionierungsverfahren relativ nahe beieinander, wobei allerdings bei wachsender Partitionsanzahl für die mittelgroße Schaltung inv10000_chain insbesondere bei Soccer aber auch bei der balancierten und Cluster bildenden Kegelpartitionierung Leistungseinbrüche zu verzeichnen sind.

Für die Schaltung inv200000_chain muß die Länge der Eingabevektoren erhöht werden, da ansonsten nur die vordersten Teile der Pipeline gefüllt werden. Das letzte Drittel der Schaltelemente wird bei eine Simulationslänge von 102,4 μs nicht aktiviert. Die Beschleunigung weicht deshalb auch von den Ergebnissen der restlichen Schaltungen ab. Deshalb wurden Messungen mit 4 und 8 Knoten für verschiedene Eingabelängen untersucht. Mit längeren Stimuli liegen die Beschleunigungswerte wiederum im Bereich der bei inv10000_chain und inv100000_chain beobachteten Ergebnisse.

Mit wachsender Modellgröße lassen sich somit bessere Leistungen erzielen. Diese Aussage deckt sich auch mit dem von Fujimoto beschriebenen Phänomen der Nachrichtenlawine [FUJ88a], nach dem die Prozessoren stets über genügend Ereignisse verfügen müssen. Erst ab einer gewissen Ereigniszahl beobachtete auch er bessere Speedups.

10.3 Skalierbarkeit des Kommunikationsnetzwerks

Während auf dem GC/PP die Kommunikation zwischen zwei Rechnerknoten für die betrachtete Paketgröße von ca. 100 Byte bei 300 μs liegt, benötigt der Versand einer solchen Nachricht im lokalen Ethernet mit 3,3 ms die 11-fache Zeit. Selbst bei abgeschalteter Konvertierung der Nachrichtendaten in das rechnerunabhängige Format XDR vergehen zwischen Versand und Empfang einer Nachricht 3 ms[4]. Die Kommunikation zwischen zwei LPs, die auf demselben Rechner plaziert werden, läuft etwas schneller ab. Sie benötigt aber dennoch 1,1 ms und ist somit knapp viermal langsamer als auf dem GC/PP. Der Aufwand für Datenkonvertierung ist hierbei nicht mehr feststellbar.

Bei den Messungen wurde nicht explizit darauf geachtet, daß sie auf "leeren" Maschinen durchgeführt wurden. Es ging hier vielmehr um die Beurteilung, welche Mindestleistung die parallelen Verfahren auf einem Produktionsrechnernetz erbringen können, in dem auch andere Benutzer arbeiten. Erst wenn sich hier merkliche Beschleunigungen einstellen, ist die Voraussetzung für einen sinnvollen und akzeptablen Einsatz paralleler Techniken in der Praxis gegeben.

10.3.1 Konservative Methoden

Es werden im folgenden ausgewählte Ergebnisse bei den Untersuchungen auf dem Workstation-Cluster wiedergegeben. Die Präsentation orientiert sich an der bereits in Kapitel 8 eingeführten Struktur.

10.3.1.1 Basisgranularität

Die sequentielle Simulationsdauer bewegt sich auf den Arbeitsplatzrechnern in derselben Größenordnung wie sie auch auf dem Parallelrechner anzutreffen ist. Allerdings ergaben sich aufgrund der nicht exklusiven Nutzung stärkere Laufzeitschwankungen, die jedoch durch das Mitteln über mehrere Simulationsläufe gedämpft werden. Der Referenzwert für die Laufzeit des sequentiellen Simulators für das Modell s13207 beträgt 64 Sekunden.

Anzahl Part.	Simulationsdauer						
	azykl.	Fan-in	Fan-out	Fan-in	Fan-out	K/L	Soccer
		unb.		bal.			
1		134,96	126,78	164,39	179,06		
2	91,77	97,05	316,34	94,60	243,83	2480,51	311,35
4	99,33	168,52	423,80	156,76	430,12	1860,17	485,32

Tabelle 10.1: Konservative Simulationsdauer, s13207

Abbildung 10.3 zeigt für die Untersuchungen auf den LAN-gekoppelten Rechnern dieselben Charakteristika wie die Kurve der Ergebnisse für den Parallelrechner. Die Leistung der paralle-

[4]Da PVM auf dem GC/PP ohnehin von einer homogenen Rechnerarchitektur ausgeht, ist dort XDR implizit abgeschaltet. Infolgedessen sind auch keine Unterschiede zwischen den Kommunikationszeiten bei ein- und ausgeschalteter XDR-Option erkennbar.

len Workstationvariante liegt hier allerdings um bis zu 50 Prozent niedriger als auf dem GC/PP. Insgesamt wurde kein absoluter Speedup im Vergleich zur sequentiellen Version des Simulators beobachtet.

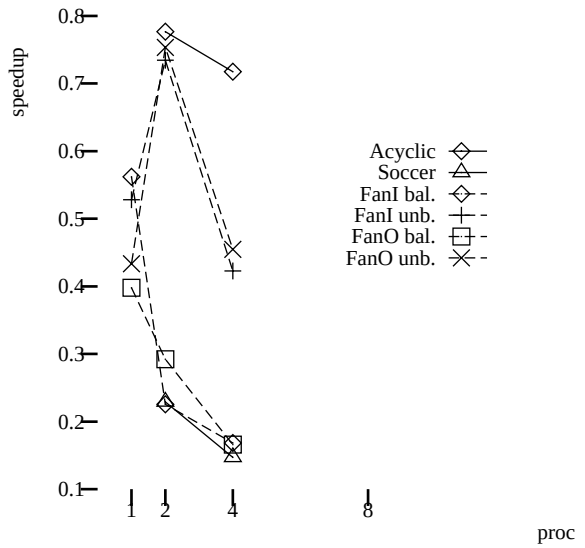


Abbildung 10.3: Speedup, s13207, Deadlock-detection

Das künstliche Modell inv10000_8_cs schnitt mit seinem zyklischen Aufbau in der Workstation-Umgebung extrem schlecht ab und lief im besten Fall siebenmal langsamer als der sequentielle Simulator.

10.3.1.2 Erhöhte Granularität

Die Werte für die erhöhten Granularitäten der Ereignisse wurden zunächst wie auf dem GC/PP mit 300 ms und 1500 ms gewählt. Unter modifizierten Kommunikationsrandbedingungen verändern sich auch die Verhältnisse von konstant gebliebener Ereignisgranularität zu Nachrichtenaufwand in den beiden betrachteten Fällen auf 1:10 bzw. 1:2. Im besten Fall ist also die Ereignisdauer immer noch geringer als die Kosten für den Versand einer Nachricht.

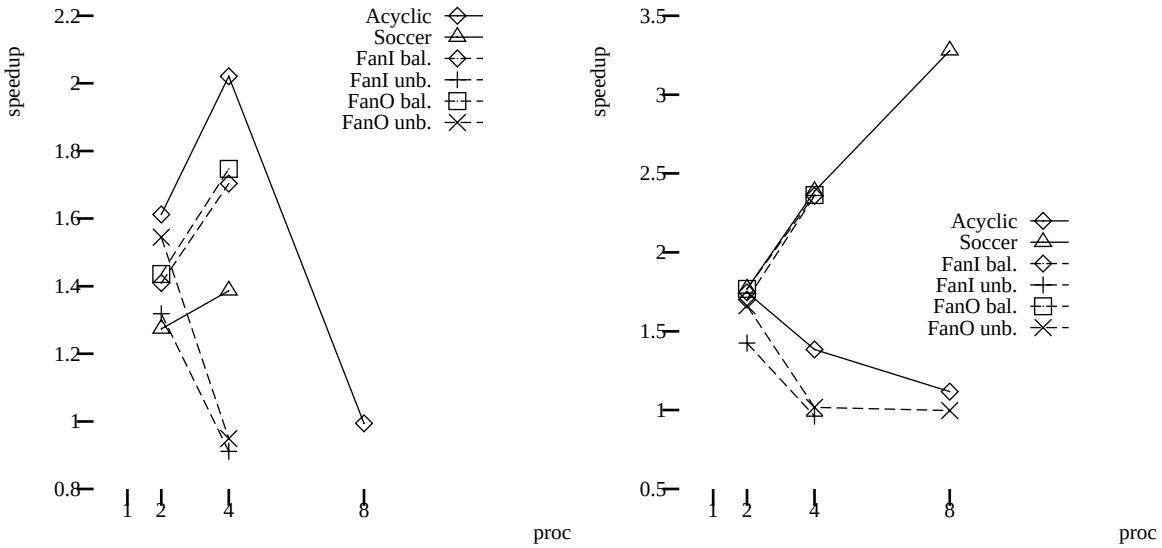
Anzahl Part.	Simulationsdauer					
	azykl.	Fan-in	Fan-out	Fan-in	Fan-out	Soccer
		unb.		bal.		
2	997,97	1220,32	1141,77	1041,19	1120,58	1263,45
4	1146,57	1765,31	944,42	1695,23	921,07	1160,64
8	1617,97					

Tabelle 10.2: Konservative Simulationsdauer bei Faktor 1:10, s13207

Auch hier zeigen sich wie bei den Untersuchungen mit Basisgranularität Parallelen zu den in Kapitel 8 gefundenen Ergebnissen. Bereits bei leicht erhöhter Granularität (die Dauer eines Nachrichtenversands entspricht der für die Bearbeitung von zehn Ereignissen) steigen die Beschleunigungen über den kritischen Wert "1". Steigert man die Ereignisgranularität weiter auf das Verhältnis 1:2, so erhält man auch auf dem Workstation-Cluster qualitativ dieselben Kurven, die mit ausgeglichener Granularität auf dem GC/PP beobachtet wurden.

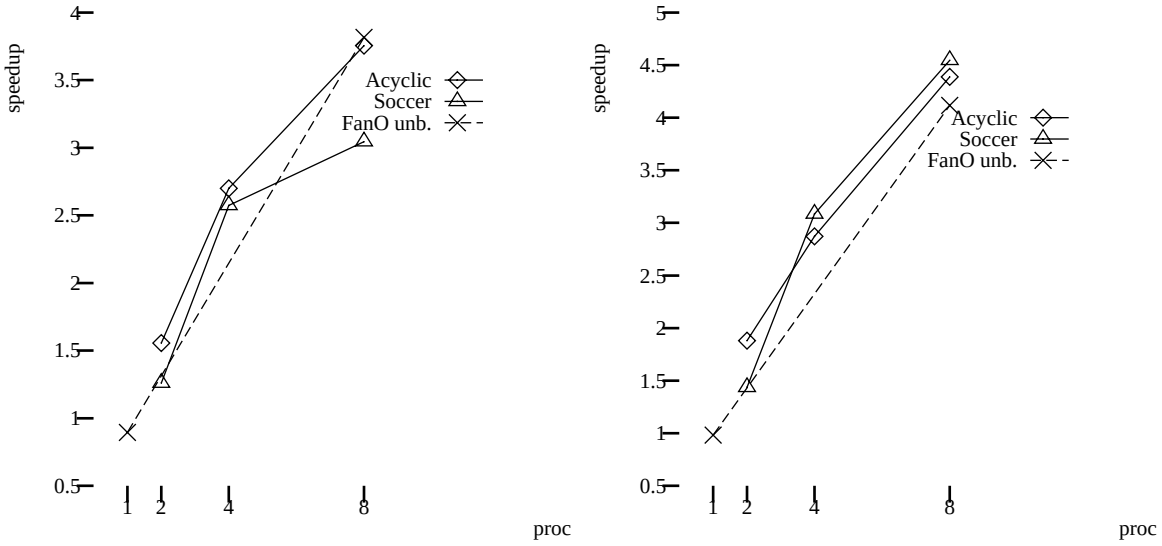
Anzahl Part.	Simulationsdauer					
	azykl.	Fan-in	Fan-out	Fan-in	Fan-out	Soccer
		unb.		bal.		
2	4979,43	6099,14	5114,34	5228,89	4915,47	4916,05
4	6275,84	9037,61	3685,95	8546,37	3677,25	3641,82
8	7777,27			8723,04		2652,22

Tabelle 10.3: Konservative Simulationsdauer bei Faktor 1:2, s13207



a) 1:10 b) 1:2

Abbildung 10.4: Speedup mit variierender Granularität, s13207, Deadlock-detection



a) 1:10 b) 1:2

Abbildung 10.5: Speedup mit variierender Granularität, s35932, Deadlock-detection

Die LAN-Variante liegt dabei nur wenig unterhalb der Speedup-Werte des Parallelrechners, obwohl das Verhältnis von Ereignis- zu Nachrichtengranularität im Workstationnetz ungünstiger ist. Diese Aussagen treffen auch für das größere reale Modell s35932 zu. Die Kurven ähneln denen der Parallelrechnervariante von DVSIM und weisen ebenfalls Beschleunigungen mit einer Effizienz von über 50 Prozent auf. Die Resultate für die modifizierten Granularitäten sind in Abb. 10.5 dargestellt.

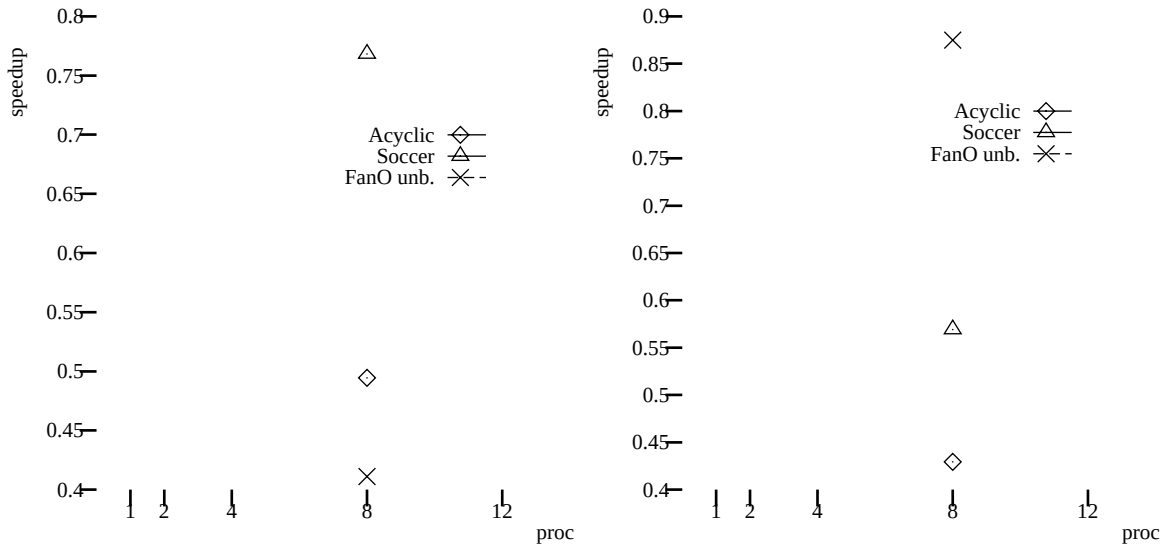
Die Erfahrungen mit zwei unterschiedlichen Rechnerplattformen sind somit für die konservativen parallelen Simulationsmethoden weitgehend deckungsgleich. Die Simulationsgeschwindigkeit nimmt mit wachsender Ereignisgranularität zu und es können durchaus interessante Beschleunigungswerte erreicht werden. Der Einfluß der Nachrichtengranularität gibt dabei quasi eine Mindestgranularität für die Ereignisse vor, die nicht unterschritten werden darf, wenn diese Beschleunigung ausgenutzt werden soll. Die hier durchgeführten Untersuchungen legen mindestens ein Gleichgewicht oder sogar ein Überwiegen des Ereignisaufwands nahe.

10.3.2 Optimistische Methoden

Analog zu den konservativen Verfahren soll nun für den Time Warp überprüft werden, ob ähnliche Aussagen auch für die Übertragbarkeit der Ergebnisse vom GC/PP auf die LAN-Version getroffen werden können.

10.3.2.1 Basisgranularität

Es stellt sich bereits bei den ersten Simulationsversuchen heraus, daß die realen Modelle sich unter Time Warp auf unserem Workstation-Cluster nicht immer vollständig simulieren lassen. Lediglich Simulationsläufe mit der kleineren Schaltung s1196 und einige wenige bei den größeren Modellen brachen nicht wegen Speichermangel ab.



a) s13207 b) s35932

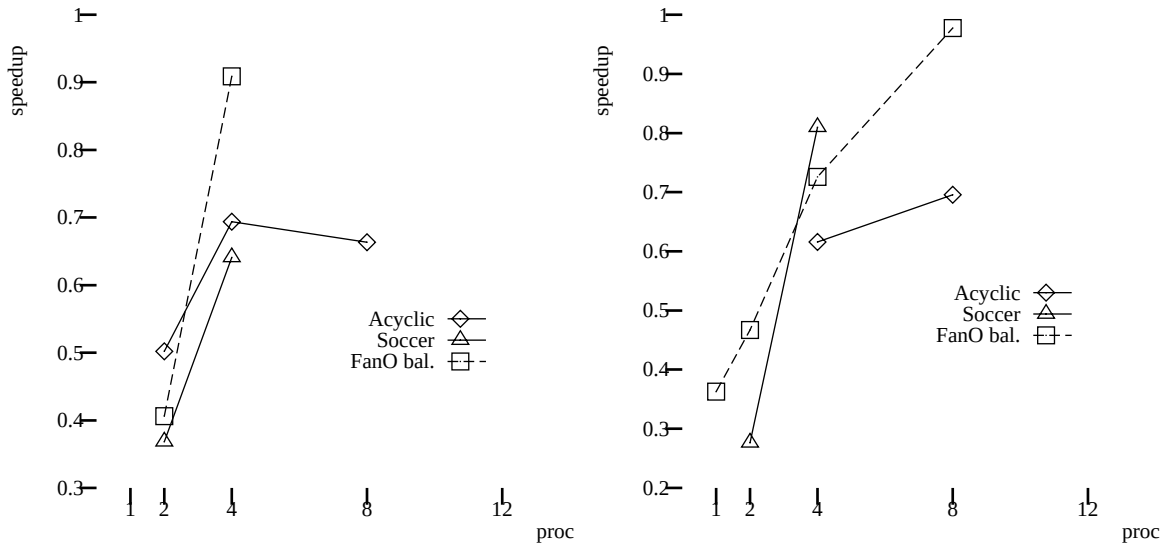
Abbildung 10.6: Time Warp mit limitiertem Optimismus

Die Ursachen dafür liegen vermutlich in der verspäteten Auslieferung der Antinachrichten aufgrund höherer Nachrichtenlaufzeiten. Dadurch werden einerseits die zu optimistischen Simu-

latoren nicht genügend gebremst und andererseits ist ein ausreichender Fortschritt der GVT-Approximation, der zu einer Freigabe nicht mehr benötigter Daten führt, nicht mehr gegeben.

Die einzelnen LPs belegen somit für die Aufbewahrung aller zwischen GVT und LVT liegenden Ereignisse sowie der empfangenen und selbst verschickten Nachrichten extrem viel Speicherplatz. Die minimal pro Prozessor vorhandenen 64 MB Hauptspeicher[5], die allerdings nicht exklusiv für die Simulation verfügbar sind, reichen überraschenderweise nicht dazu aus, diese Daten aufzunehmen. Die reine Hauptspeicherkonfiguration auf dem GC/PP war vergleichbar. Dort konnten auch ohne Einsatz von Swapbereichen wesentlich mehr Simulationsläufe erfolgreich beendet werden.

Wurde dagegen der Optimismus durch dynamische Fenstertechniken beschränkt, so lieferten die Schaltungen für die initiale Fenstergröße "1000" die in Abb. 10.6 dargestellten Ergebnisse. Absolute Beschleunigungswerte konnten allerdings auch hiermit nicht erreicht werden.



a) s13207 b) s35932

Abbildung 10.7: Time Warp mit Aggressive-cancellation

Verwendet man dagegen die aggressive Version des Time Warp, so werden zwar keine absoluten Beschleunigungen damit erreicht, aber die meisten Simulationen laufen auch ohne Einsatz von Fensterverfahren erfolgreich bis zum Ende durch (Abb. 10.7). Dieses Verhalten ist ein weiteres Indiz dafür, daß der übermäßige Speicherbedarf durch den ungezügelten Optimismus der LC-Version verstärkt wird. Die Kombination von Fenstertechniken und Aggressive-cancellation brachte ebenfalls keine besseren Ergebnisse (Abb. 10.8).

Bei den synthetischen Schaltungen wurden Messungen mit inv10000_chain und inv10000_8_cs durchgeführt. In der Mehrzahl der Fälle brachen auch hier die Simulationen mit dem optimistischen Time Warp aufgrund von Speichermangel ab. Bei den wenigen erfolgreichen Experimenten auf zwei Knoten für die Schaltung inv10000_chain benötigte die parallele Version etwa 50 Prozent mehr Zeit als die sequentielle Variante. Die parallele Simulation auf einem einzigen Knoten war sogar dreimal langsamer.

Das zyklisch organisierte Modell inv10000_8_cs zeigte jedoch bei Verwendung von 8 Prozessoren für die balancierte Fan-in-Kegelpartitionierung sogar einen echten Speedup von 1,2. Aufgrund der vielen Rückkopplungen wurde hier vermutlich der Optimismus des Time Warp stärker gebremst als

[5] und die zusätzlichen Auslagerungsbereiche im Swap-space, die nochmals ebenso groß sind.

beim quasi unabhängig bearbeitbaren Pipeline-Modell. Die Verwendung der zusätzlich bremsenden aggressiven Time Warp-Variante brachte wie bei den realen Modellen ebenfalls keine weiteren Verbesserungen.

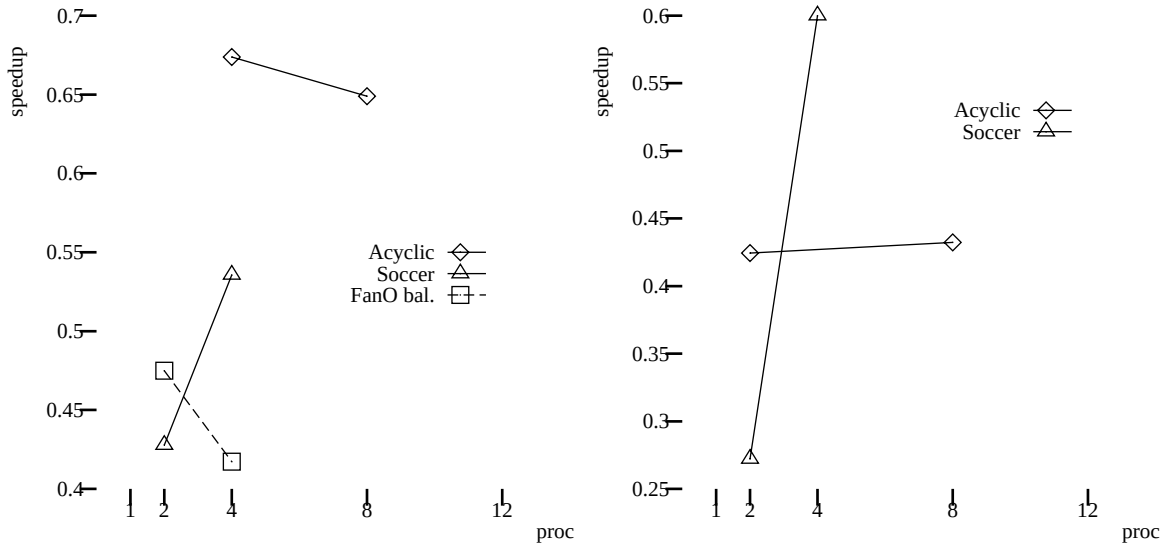


Abbildung 10.8: Time Warp mit limitiertem Optimismus und Aggressive-cancellation

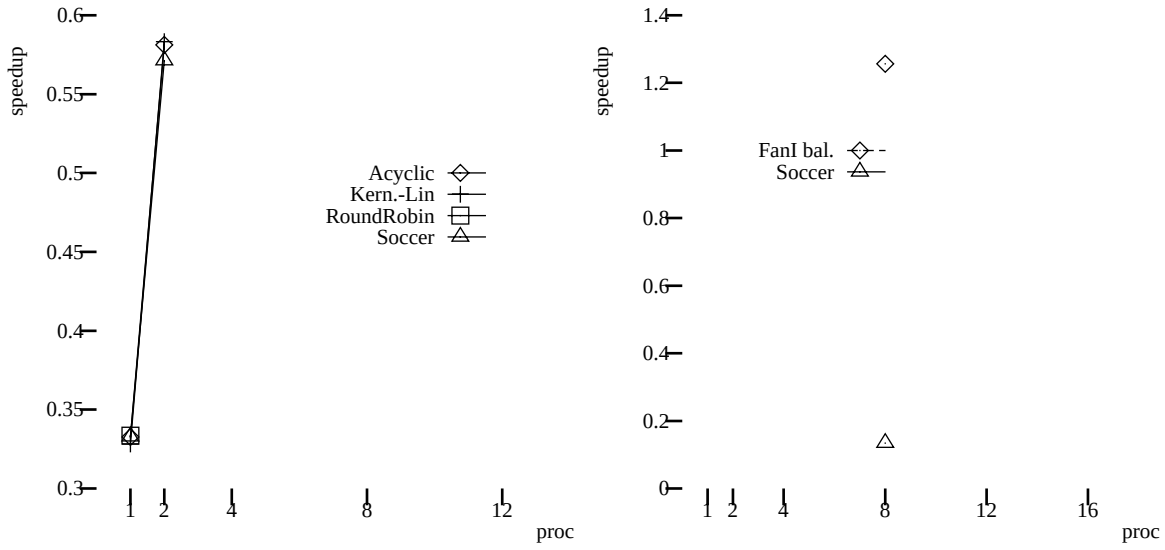


Abbildung 10.9: Time Warp, Lazy-cancellation

10.3.2.2 Erhöhte Granularität

Auch bei den optimistischen Simulationsversuchen mit erhöhter Granularität ergaben sich für die realen Modelle Probleme mit dem zur Verfügung stehenden Speicher. Ohne Limitierung des Optimismus konnten nur wenige Simulationsläufe ordnungsgemäß zu Ende geführt werden.

Einige der wenigen Fälle, bei denen die Simulation ohne Abbruch bis zum Ende lief, stellt Modell s35932 mit dem Granularitätsverhältnis 1:2 und dem Soccer-Verfahren auf 8 Prozessoren dar. Es zeigt sich ein interessantes Parallelisierungspotential von 6,2, obwohl die Kosten des Nachrichtenversands immer noch doppelt so hoch sind, wie die Dauer einer Ereignisroutine (Abb. 10.10). Dieses Ergebnis liegt um einiges höher als die Beschleunigung des konservativen Verfahrens mit identischen Parametern.

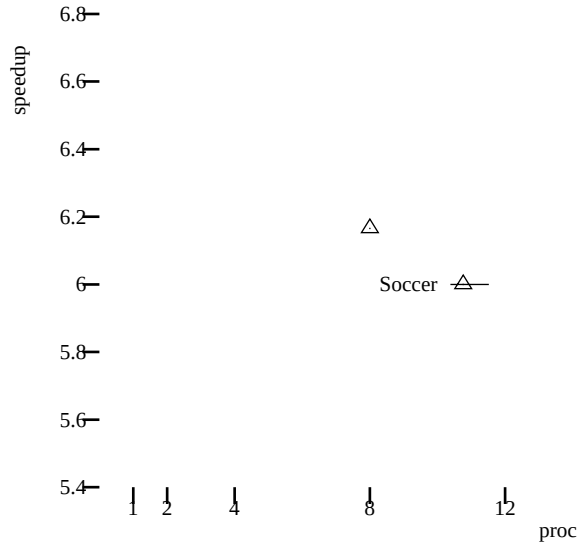


Abbildung 10.10: Speedup unter Time Warp mit Granularität 1:2, s35932

Bei Erhöhung der Granularität auf 1:2 für das künstliche Modell inv10000_chain wurden mit Lazy-cancellation und einer Limitierung des Optimismus mit dynamischen Anpassung der Fenstergröße ebenfalls Speedups von 5,3 auf 8 Prozessoren erreicht.

Insgesamt ergibt sich also für die Workstationvariante des parallelen Simulators DVSIM ein tendenziell ähnliches Bild wie auf dem Parallelrechner GC/PP. Unter erhöhter Granularität der Ereignisse werden auch auf einem konventionellen Cluster von Arbeitsplatzrechnern Beschleunigungen erzielt. Die konservativen Verfahren laufen dabei sehr stabil. Die optimistischen Varianten benötigen zusätzliche Mechanismen wie Limitierung des Optimismus durch Fenstertechniken oder sofortigen aggressiven Versand der Nachrichten bei einem Rollback. Andernfalls brechen die Simulationen vorzeitig mit Speichermangel ab. Auch dieses Verhalten ist auf einem Workstationcluster wesentliche ausgeprägter als auf dem Parallelrechner.

Kapitel 11

Zusammenfassung und Ausblick

In einer breit angelegten Reihe von Untersuchungen mit einer prototypischen Implementierung eines parallelen ereignisgesteuerten Simulators für die Schaltkreissimulation auf Gatterebene wurden in dieser Arbeit alle relevanten Aspekte, die in den vergangen zwei Jahrzehnten oft nur als einzelne Details bewertet wurden, in einen umfassenden gemeinsamen Kontext gestellt. Dadurch war es möglich, wichtige Faktoren zu identifizieren und eine gesicherte Beurteilung ihres Einflusses auf das Parallelisierungspotential dieser Applikationsklasse vorzunehmen. Bei der Realisierung des parallelen Simulators wurde auf effiziente Implementierung geachtet und als Referenz für die Leistungsbewertung ein optimierter sequentieller Simulator eingesetzt. Die Verwendung der Laufzeiten des parallelen Simulationsalgorithmus auf lediglich einem einzigen Knoten führte in anderen Untersuchungen oft zu überhöhten Beschleunigungswerten und ist deshalb als unfair abzulehnen. An dieser Stelle soll abschließend der Bogen über die Erkenntnisse des Evaluierungsprozesses geschlagen werden.

11.1 Bewertung der Untersuchungen zur Logiksimulation

Zunächst wurde im Rahmen dieser Arbeit versucht, anhand der Kritische-Pfad-Analyse eine Voraussage über das inhärente Beschleunigungspotential der Schaltkreissimulation bei unbeschränkten Ressourcen zu treffen. Dabei wurde festgestellt, daß diese idealisierte Annahme ebenso wie die Vorstellung, daß Kommunikation keine Rolle spielt, zu einer extremen Überschätzung des tatsächlich vorhandenen Potentials führt. Durch Einführung limitierter Prozessoranzahlen mit einer vorgegebenen Partitionierung sowie konstanter Kommunikationszeiten für die Kritische-Pfad-Analyse konnten vernünftige Vorhersagen, die das Verhalten der Schaltungen bei echter Parallelsimulation treffender beschreiben, gemacht werden.

In diesem Zusammenhang ist auch festzustellen, daß die Partitionsgröße und die Aufteilung der Schaltung eine sehr große Bedeutung für die Leistungsfähigkeit der Simulation besitzt. Dabei gilt es zwischen konservativen und optimistischen Verfahren zu unterscheiden. Bei den konservativen Synchronisationsalgorithmen sind in erster Linie Blockaden zu vermeiden, die zu aufwendigen Deadlockerkennungs- und auflösungsaktionen führen. Partitionierungen, die Zyklen vermeiden, und dadurch unter Umständen unbalancierte Teilmodelle generieren oder auch die Topologie stark berücksichtigen (z.B. unbalancierte Kegelpartitionierung), zeigen wesentlich bessere Ergebnisse als Heuristiken, die die Schaltung auf einen allgemeinen Graphen abbilden und versuchen, die Schnittkosten zu minimieren (z.B. Soccer, K/L).

Bei den optimistischen Methoden schneiden dagegen die unbalancierten Partitionen schlecht ab. Hier kommt es eher darauf an, ausgewogene Partitionen zu verwenden. Die balancierten Ke-

gelpartitionierungen sind in diesem Fall den anderen Verfahren überlegen. Allerdings fallen die Unterschiede zum allgemeineren Soccer-Verfahren, das auch in vielen Fällen recht gute Ergebnisse liefert, nicht so stark aus.

Daneben kommt es auch noch auf eine einzuhaltende Mindestgröße der Partitionen an. Unterschreiten Teilmodelle eine bestimmte Größe (in unserem Fall wenigstens 3000 Gatter), so sind die Simulatoren oft nicht mehr ausgelastet und die Effizienz der Beschleunigungswerte bricht stark ein.

Auch hinsichtlich der Leistungsfähigkeit der verschiedenen Synchronisationsalgorithmen konnten, im Gegensatz zu vielen Berichten in der Literatur, die optimistischen Methoden nicht eindeutig überzeugen. Möglicherweise liegt das an den in vielen anderen Arbeiten fälschlicherweise zugrundegelegten Basiswerte für den Speedup, die sehr oft auf der Einprozessor-Version der parallelen Simulatoren beruhen. Wie von uns beobachtet, generiert diese unter Time Warp relativ zur sequentiellen Simulation wesentlich mehr Aufwand als eine konservative Version. Folglicherweise fallen scheinbar auch die Beschleunigungen beim Vergleich wesentlich besser aus.

Insgesamt zeigten jedoch die untersuchten konservativen Verfahren ein stabileres Verhalten bezüglich der erfolgreichen Durchführbarkeit von Simulationsläufen für die verschiedenen Schaltkreise. Der Time Warp wies oft die entsprechend der Theorie zu erwartenden Speicherprobleme auf, so daß viele Simulationen mit dem Basisverfahren (Lazy-cancellation) aufgrund von Speichermangel abbrachen. Die vorgeschlagenen Alternativen wie Aggressive-cancellation und Fenstertechniken lieferten in Fällen, in denen alle Varianten erfolgreich durchliefen, schlechtere Ergebnisse als Lazy-cancellation. Speicherknappheit trat bei den beiden modifizierten Verfahren jedoch weitaus seltener auf.

Weiterhin wurde auch der zusätzliche Aufwand für die initiale Partitionierung, die Verteilung der Schaltung und die Terminierung der verteilten Algorithmen untersucht. Durch Berücksichtigung dieser Kosten sinken die ohnehin schon relativ bescheidenen Beschleunigungswerte, die die parallele Simulation lieferte, noch weiter ab, so daß insbesondere durch die Partitionierungskosten die reine Laufzeitbeschleunigung teilweise vollständig zunichte gemacht wird.

Insgesamt läßt sich aus den Untersuchungen folgern, daß sich die parallele Simulation einer niedrigen Granularitätsstufe (Gatterebene) im allgemeinen kaum lohnt. Ursache dafür ist vor allem die geringe Granularität der Ereignisse. Das Verhältnis zu den Nachrichtenkosten ist bereits auf einem Parallelrechner sehr schlecht und klafft auf einem Workstation-Cluster mit einem lokalen Netz noch wesentlich stärker auseinander. Aus diesem Grund kann die ereignisgesteuerte Parallelsimulation wohl nicht als ein universell einsetzbarer und einfach realisierbarer Weg zu kürzeren Simulationszeiten bei dieser Applikationsklasse betrachtet werden.

11.2 Auswirkungen auf das allgemeine Gebiet der parallelen Simulation

Die Aussagen, die für die Schaltkreissimulation getroffen wurden, lassen sich bis auf wenige Ausnahmen auch auf sonstige Applikationsklassen übertragen. Die gilt insbesondere für die Nebenkosten der Simulation und die Bedeutung der Partitionierung. Allerdings gibt es eine wichtige Größe, die hier einen wesentlichen Unterschied bei der Gesamtbewertung ausmacht. Bei den Untersuchungen, die mit einer erhöhten Granularität der Ereignisse durchgeführt wurden, ergaben sich wesentliche Verbesserungen bei der Leistung der parallelen Verfahren im Vergleich zu den sequentiellen Simulationsläufen. Erreicht die Bearbeitungsdauer der Ereignisse eine gewisse kritische Grenze, so steigen die Beschleunigungswerte bei den beobachteten Schaltungen drastisch an. Es wurden teilweise Effizienzwerte von über 70 Prozent beobachtet.

Weist also eine Applikationsklasse eine entsprechend hohe Granularität der Ereignisse auf und besitzen die Partitionen die geforderten Mindestgrößen, so sind mit hoher Wahrscheinlichkeit interessante Beschleunigungswerte zu erwarten. Außerdem verlieren durch die damit i.a. einhergehenden längeren Laufzeiten auch die Nebenkosten an Gewicht.

Insbesondere auf dem Workstation-Cluster konnten bei künstlich verlängerten Ereignisroutinen Beschleunigungen beobachtet werden. Die konservativen Verfahren schnitten wesentlich besser ab als der Time Warp, dessen Speicherbedarf sich hier noch störender auswirkte als auf dem verwendeten Parallelrechner.

Die Frage, ob durch schnellere Kommunikationsnetze, Prozessoren oder mehr Speicher gute Ergebnisse erreicht werden könnten, läßt sich nicht definitiv beantworten. Vielmehr liegt aufgrund der durchgeführten Untersuchungen und der Beobachtungen der letzten zwanzig Jahre, in denen diese Faktoren stets verbessert wurden, die Vermutung nahe, daß diese Forderungen, mit denen viele frühere Veröffentlichungen endeten, nicht so maßgeblich sind, da dadurch ja auch die Vergleichsbasis der sequentiellen Simulation beeinflußt wird. Auch der Einsatz neuer Algorithmen hat oft nur marginale Verbesserungen gebracht. Die Komplexität und Heterogenität der Anwendungsklasse scheint, sofern keine ausreichende Granularität vorhanden ist, der Beschleunigung inhärent im Wege zu stehen.

11.3 Parallele Simulation — ein universelles Tool?

Neben diesen eigentlichen technischen Aspekten stellen sich jedoch noch weitere Probleme dem allgemeinen Einsatz universeller paralleler Simulatoren in den Weg. Zum Beispiel ist heute ein Anwender an Simulationsumgebungen und -werkzeuge gewöhnt, die er auch bei Verwendung eines parallelen Tools erwartet. Leider gibt es nur wenige Ansätze dazu, wie der Utilarian-parallel-simulator (U.P.S.) von Heidelberger und Nicol [HEN96a] oder Vorschläge zur transparenten Behandlung von Zustandssicherungen im Time Warp [RLA96a], [WEP96a], um die sich ein Anwender verständlicherweise auch nicht kümmern möchte.

Ein universell einsetzbarer paralleler Simulator steht natürlich auch in gewissem Widerspruch zur Notwendigkeit des applikationsspezifischen Tunings der parallelen Algorithmen. Diese kontroverse Diskussion fand in einem Themenheft des ORSA-Journals [ORS93a] bereits 1993 statt. Seit diesem Zeitpunkt hat sich jedoch nichts wesentliches in dieser Hinsicht verändert. Die parallele Simulation steht somit im Zwiespalt zwischen höherer Geschwindigkeit und einfacher allgemeiner Verwendbarkeit.

Fujimoto thematisierte einige der Diskrepanzen und benannte notwendige Arbeitsbereiche, in denen Handlungsbedarf besteht [FUJ93b]: parallele Simulationssprachen, Simulationsbibliotheken zur Unterstützung bestimmter Modelle auf parallelen Architekturen, gemeinsame globale Variablen zur leichteren Übertragbarkeit der sequentiellen Modelle und automatische Parallelisierbarkeit (z.B. Zugriff auf Ereignislisten in gemeinsam genutztem Speicher). Verschiedene andere Autoren kommentieren diese Punkte. Jeder von ihnen geht aber leider in seiner Stellungnahme lediglich auf ganz bestimmte Details ein, die in Kapitel 3.6.1 aufgezählt wurden. Eine umfassende Behandlung der Gesamtproblematik findet sich nur in Fujimotos Thesenpapier und seinem nachfolgenden Resümee der Beiträge [FUJ93c].

Sein Fazit aus diesen Beiträgen war, daß es noch viel Aufwand bereite, um die parallele Simulation zu einer allgemein akzeptierten Technologie zu machen. Allerdings ist seit etwa 1995 zu beobachten, daß sich viele Forscher dem Bereich der Distributed-interactive-simulation (DIS) zuwenden, in dem vorwiegend die traditionellen Modelle der verteilten Simulation unter Vernachlässigung von kausalen Abhängigkeiten eingesetzt werden [FUJ95a].

Zusammenfassend wage ich die These, daß die parallele Simulation aufgrund der Komplexität zur Entwicklung geeigneter Tools auf absehbare Zeit kein massives Beschleunigungspotential für sequentielle Simulationsläufe bei heterogenen Problemklassen darstellen wird. Sie ist bei den aktuellen Trends vielmehr eine Nischenlösung für Spezialisten, die geeignete, tief in das System eingreifende Optimierungen, die zudem für jede Applikationsklasse und innerhalb einer Klasse auch für verschiedene Modelle unterschiedlich sind, erfordert. Beschleunigungsfaktoren zwischen 25 und 50 Prozent der idealen, linearen Speedup-Werte sind dann teilweise möglich. Die Bereiche der homogenen Problemstellungen mit geringer Koppelungsrate werden hier allerdings ausgenommen. Durch Datenparallelismus lassen sich mittels synchroner Koordination, z.B. über Phasen und Barrieren, bessere Ergebnisse erzielen.

Als "Nebenprodukt" der langjährigen Forschungen wurden jedoch im Bereich der parallelen und verteilten Algorithmen viele grundlegende Probleme erkannt und Lösungen dafür angegeben (Scheduling, Synchronisation, Terminierungsalgorithmen, Schnappschußverfahren). Diese Ergebnisse sind über das Feld der parallelen Simulation hinaus vielfältig einsetzbar. Auch wenn die obige Aussage zum Beschleunigungspotential paralleler diskreter ereignisgesteuerter Simulation nicht sonderlich zufriedenstellt, so sind die in den vergangenen 20 Jahren durchgeführten Arbeiten für das gesamte Feld der Informatik von großer Bedeutung.

Die Zuwendung vieler Simulationsforscher zu verteilter interaktiver Simulation (DIS) und webbasierter Simulation deutet darauf hin, daß die Forschung im Kernbereich der parallelen Simulation wohl langsam auslaufen wird und die Interessen in andere Richtungen schwenken. Vielleicht wird die Entwicklung und Integration geeigneter Tools, mit denen moderate Beschleunigungen erzielt werden können, der parallelen Simulation eine kleinere Rolle für Randprobleme zuweisen. Die Antwort auf diese Frage ist allerdings alles andere als gewiß. Die Lockerung der Restriktionen bezüglich kausaler Abhängigkeiten, wie sie in der verteilten interaktiven Simulation vorhanden ist, stellt jedenfalls für viele Applikationen keine praktikable Lösung dar. Die Verwendung von Simulatorfrontends, die über das Internet für eine große Nutzergruppe angeboten wird, könnte durch eine für den Benutzer transparente Ankopplung eines Parallelrechners oder Workstation-Clusters eine weitere mögliche Anwendung der parallelen Simulation (ebenfalls unter herabgesetzten Erwartungen an die Leistungsfähigkeit) darstellen. Der Nutzer einer solchen Schnittstelle erfährt auf diese Weise gar nicht, daß dahinter eine komplexe, von Spezialisten entwickelte und hoch optimierte Software steckt. Er nutzt lediglich die nach außen angebotene Sicht als Simulationsdienst, der auf bereitgestellte Modelle und Eingaben die entsprechenden Resultate liefert.

Literaturverzeichnis

[ABC88a] Alverson, B., Blank, T., Choi, K., Hwang, S., Salz, A., Soulé, L., und Rokicki, T. (1988), THOR User's Manual: Tutorial and Commands, Technical Report CSL-TR-88-348, Stanford University

[ABR93a] Abrams, M. (1993), *Parallel Discrete Event Simulation: Fact or Fiction?* ORSA Journal on Computing, Vol. 5, Nr. 3, S. 231 – 233

[ADM93a] Ashenden, P. J., Detmold, H., und McKeen, W. S. (1993), *Parallel Execution of VHDL Models.* Technical Report 93-01, University of Adelaide

[APW93a] Aji, S., Palaniswamy, A., und Wilsey, P. (1993), *Interactions of Optimizations to a Time Warp Synchronized Digital System Simulator.* Proc. of the European Simulation Multiconference, S. 593 – 597

[ARS91a] Arvind, D. und Smart, C. (1991), *A Unified Framework for Parallel Event-Driven Logic Simulation.* Technical Report, Department of Computer Science, University of Edinburgh, Scotland

[AVT95a] Avril, H. und Tropper, C. (1995), *Clustered Time Warp and Logic Simulation.* Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS), Vol. 25, S. 20 – 27, http://www.acm.org/pubs/citations/proceedings/simulation/214282/p112-avril/

[AVT96a] Avril, H. und Tropper, C. (1996), *The Dynamic Load Balancing of Clustered Timewarp for Logic Simulation.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 20 – 27, http://www.acm.org/pubs/citations/proceedings/simulation/238788/p20-avril/

[BAG93a] Bagrodia, R. (1993), *A Survival Guide for Parallel Simulation.* ORSA Journal on Computing, Vol. 5, Nr. 3, S. 234 – 235

[BAG96a] Bagrodia, R. L. (1996), *Perils and Pitfalls of Parallel Discrete-Event Simulation.* Proceedings of the 1996 Winter Simulation Conference, S. 136 – 143, ftp://pcl.cs.ucla.edu/pub/papers/wsc96-perils.ps.gz

[BAI92a] Bailey, M. (1992), *How Circuit Size Affects Parallelism.* IEEE Trans. on Computer-Aided Design, Vol. 11, Nr. 2, S. 208 – 215

[BAJ96a] Bagrodia, R. und Jha, V. (1996), *A Performance Evaluation Methodology for Parallel Simulation Protocols.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 180 – 185

[BAS88a] Bailey, M. und Snyder, L. (1988), *An empirical study of on-chip parallelism.* Proc. of the 25th DAC, ACM/IEEE, S. 160 – 165

[BAS93a] Bauer, H. und Sporrer, C. (1993), *Reducing Rollback Overhead in Time Warp Based Distributed Simulation with Optimized Incremental State Saving.* 26th Annual Simulation Symposium ASS 93, Washington

[BAS93b] Sporrer, C. und Bauer, H. (1993), *Corolla Partitioning for Distributed Logic simulation of VLSI-Circuits.* Proc. of the 7th workshop on Parallel and Distributed Simulation, Vol. 23, Nr. 1, S. 85 – 92

[BBB87a] Bryant, R., Beatty, D., Brace, K., Cho, K., und Sheffler, T. (1987), *COSMOS: A Compiled Simulator for MOS Digital Circuits.* Proc. of the 24th Design Automation Conference, ACM/IEEE, S. 9 – 16

[BBC94a] Bailey, M., Briner Jr., J., und Chamberlain, R. (1994), *Parallel Logic Simulation of VLSI Systems.* ACM Computing Syrveys, Vol. 26, Nr. 3, S. 255 – 294

[BBC94b] Beaumont, C., Boronat, P., Champeau, J., Filloque, J.-M., und Pottier, B. (1994), *Reconfigurable Technology: An Innovative Solution for Parallel Discrete Event Simulation Support.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 160 – 163, http://www.acm.org/pubs/citations/proceedings/simulation/182478/p160-beaumont/

[BBK89a] Brglez, F., Bryan, D., und Kozminski, K., (1989), *Combinational Profiles of Sequenctial Circuits.* In Proc. of the Intl. Symposium on Circuits and Systems (ISCAS), S. 1929 – 1934

[BCC96a] Blackford, L., Choi, J., Cleary, A., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, G., Henry, G., Petitet, K., Stanley, K., Walker, D., und Whaley, R. (1996), *ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance.* Proc. of Supercomputing

[BCJ95a] Bagrodia, R., Chen, Y., Jha, V., und Sonpar, N. (1995) , *Parallel Gate-Level Circuit Simulation on Shared Memory Architectures.* Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS), Vol. 25, S. 170 – 174

[BCL91a] Bagrodia, R., Chandy, K., und Laio, W. (1991), *A Unifying Framework for Distributed Simulation.* ACM Transactions on Modeling and Computer Simulation, Vol. 1, Nr. 4, S. 348 – 385

[BDM73a] Birtwistle, G., Dahl, O., Myrhaug, B., und Nygaard, K. (1973), *Simula Begin.* Auerbach Publishers Inc

[BEJ85a] Berry, O. und Jefferson, D. (1985), *Critical Path Analysis of Distributed Simulation.* Proceedings of the Distributed Simulation Conference, San Diego, S. 57 – 60

[BEK91a] Briner, Jr., J., Ellis, L. und Kedem, G. (1991), *Breaking the Barrier of Parallel Simulation of Digital Systems.* Proc. of the 28th DAC, ACM/IEEE, S. 223 – 226

[BEL90b] Bemmerl, T. und Ludwig, T. (1990), *MMK – A Distributed Operation System Kernel with Integrated Loadbalancing.* Proc. of the CONPAR 90 VAPP IV, LNCS, Vol. 457, S. 744 – 755

[BJW94a] Bagrodia, R. L., Jha, V., und Waldorf, J. (1994), *The Maisie Environment for Parallel Simulation.* Proceedings of the 1994 Annual Simulation Symposium

[BLJ94a] Bagrodia, R. L., Li, Z., Jha, V., Chen, Y.-A., und Cong, J. (1994), *Parallel Logic Level Simulation of VLSI Circuits.* Proceedings of the 1994 Winter Simulation Conference, S. 1354 – 1361

[BLU90a] Baezner, D., Lomow, G., und Unger, B. (1990), *Sim++: The Transistion to Distributed Simulation.* SCS Multiconference on Distributed Simulation, Vol. 22, S. 211 – 218

[BOD97a] Boukerche, A. und Das, S. K. (1997), *Dynamic Load Balancing Strategies for Conservative Parallel Simulations.* Proc. of the 11th Workshop on Parallel and Distributed Simulation (PADS), Vol. 27, S. 20 – 28, http://www.computer.org/conferen/proceed/7964abs.htm#E37E7

[BOD98a] Boukerche, A. und Das, S. L. (1998), *Load Balancing Strategies for Parallel Simulations on A Multiprocessor Machine.* In Bagchi, K., Walrand, J., und Zobrist, G. W., Hrsg., Advanced Computer Performance Modeling and Simulation, S. 135 – 164

[BRF85a] Brglez, F. und Fujiwara, H. (1985), *A Neutral Netlist of 10 Combinational Benchmark Circuits and A Target Translator in FORTRAN.* In Proc. of the Intl. Symposium on Circuits and Systems (ISCAS)

[BRI90a] Briner, J. (1990), *Parallel Mixed-Level Simulation of Digital Circuits Using Virtual Time.* Technical Report TR90-38, Duke University, Durham, NC 27706

[BRY77a] Bryant, R. (1977), *Simulation of Packet Communication Architecture Computer Systems.* Internal Report, TR-188, MIT, Cambridge, Massachussetts

[BRY79a] Bryant, R. (1979), *Simulation on a Distributed System.* Proceedings of the 1st International Conference on Distributed Computer Systems, S. 544 – 552

[BSK91a] Bauer, H., Sporrer, C., und Krodel, T. H. (1991), *On distributed Simulation Using Time Warp.* In Proc. of VLSI '91, S. 127 – 136

[BSK92a] Bauer, H., Sporrer, C., und Krodel, T. (1992), *On Distributed Simulation Using Time Warp.* TR TUM-I9212, SFB Bericht 342/9/92a, Technische Universität München

[BUL92a] Butler, R. und Lusk, E. (1992), *User's Guide to the P4 Programming System.* Technical Report ANL-92/17, Argonne National Laboratory

[BUM93a] Burdorf, C. und Marti, J. (1993), *Load Balancing Strategies for Time Warp on Multi-User Workstations.* Computer Journal, Vol. 36, Nr. 2, S. 168 – 176

[CAF96a] Carothers, C. D. und Fujimoto, R. M. (1996), *Background Execution of Time Warp Programs.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 12 – 19 http://www.cc.gatech.edu/grads/c/Chris.Carothers/PAPERS/pads-96.ps

[CAP94a] Cap, C. H. (1994), *Workstation Cluster Computing aus der Sicht des Anwenders.* Praxis der Informationsverarbeitung und Kommunikation, Vol. 17, Nr. 4, S. 230 – 237

[CCG97a] Cloutier, J., Cerny, E., und Guertin, F. (1997), *Model Partitioning and ther Performance of Distributed Timewarp Simulation of Logic Circuits.* Simulation Practice and Theory, Vol. 5, S. 83 – 99, http://www.elsevier.nl:80/inca/publications/store/5/2/4/1/6/5/ ?menu=cont&label=table

[CDF94a] Costa, A., De Gloria, A., Faraboschi, P., und Oliveri, M. (1994), *An Evaluation System for Distributed-Time VHDL Simulation.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 147 – 150, http://www.acm.org/pubs/citations/proceedings/simulation/182478/p147-costa/

[CFE94a] Carothers, C. D., Fujimoto, R. M., und England, P. (1994), *The Effect of Communication Overheads on Time Warp Performance: An Experimential Study,.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), S. 118 – 125, http://www.cc.gatech.edu/grads/c/Chris.Carothers/PAPERS/pads-94.ps

[CGU94a] Cleary, J. G., Gomes, F., Unger, B. W., Zhonge, X., und Thudt, R. (1994), *Cost of State Saving & Rollback.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 94 – 100, http://www.acm.org/pubs/citations/proceedings/simulation/182478/p94-cleary/

[CHA85a] Chamberlain, R. (1985), *Lsim: A Gate-Switch Level Logic Simulator.* Master's thesis, Department of Computer Science, Washington University, St. Louis, MO

[CHA95a] Chamberlain, R. D. (1995), *Parallel Logic Simulation of VLSI Systems.* Proc. of the 32th DAC, ACM/IEEE, S. 139 – 143, http://www.acm.org/pubs/articles/proceedings/dac/217474/p139-chamberlain/p139-chamberlain.pdf

[CHB83a] Chandak, A. und Browne, J. (1983), *Vectorization of Discrete Event Simulation.* Proceedings of the International Conference on Parallel Processing, S. 359 – 361

[CHH94a] Chamberlain, R. und Henderson, C. (1994), *Evaluation the Use of Pre-Simulation in VLSI Circuit Partitioni ng.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), S. 139 – 146, http://www.acm.org/pubs/citations/proceedings/simulation/182478/p139-chamberlain/

[CHL85a] Chandy, K. und Lamport, L. (1985), *Distributed Snapshots: Determining Global States of Distributed Systems.* ACM Transactions on Computer Systems, Vol. 3, Nr. 1, S. 63 – 75

[CHM78a] Chandy, K. und Misra, J. (1978), *A Non-Trivial Example of Concurrent Processing: Distributed Simulation.* Proceedings of COMPSAC, S. 822 – 826

[CHM79a] Chandy, K. und Misra, J. (1979), *Distributed Simulation: A case study in design and verification of distributed programs.* IEEE Transactions on Software Engineering, Vol. SE-5, Nr. 5, S. 440 – 452

[CHM81a] Chandy, K. und Misra, J. (1981), *Asynchronous Distributed Simulation Via a Sequence of Parallel Computations.* CACM, Vol. 24, Nr. 4, S. 198 – 205

[CHS89b] Chandy, K. und Sherman, R. (1989), *Space-Time and Simulation.* Proceedings of the SCS Multiconference on Distributed Simulation, S. 53 – 57

[CHW91a] Chawla, P. und Wilsey, P. (1991), *Synchronizing Distributed VHDL Simulation.* Technical Report TR 131-4-91-ECE, University of Cincinnati

[CLB94a] Cong, J., Li, Z., und Bagrodia, R. L. (1994), *Acyclic Multi-Way Partitioning of Boolean Networks.* Proc. 31st Design Automation Conference, S. 670 – 675

[CM2] Thinking Machine Corp. (1999), *Connection Machine CM-2.* http://www.connmach.com

[COE89a] Coehlo, D. R. (1989), *The VHDL Handbook.* Kluwer Academic Publishers, Norwell, Dordrecht

[COM84a] Comfort, J. (1984), *The Simulation of a Master-Slave Event Set Processor.* Simulation 42, S. 117 – 124

[DAF93a] Das, S. R. und Fujimoto, R. M. (1993), *A Performance Study of the Cancelback Protocol for Time Warp.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 135 – 142

[DAF97a] Das, S. R. und Fujimoto, R. M. (1997), *Adaptive Memory Management and Optimism Control in Time Warp.* ACM Transactions on Modelling and Computer Simulation, Vol. 7, Nr. 2, S. 239 – 271, http://www.acm.org/pubs/citations/journals/tomacs/1997-7-2/p239-das/

[DIR90a] Dickens, P. und Reynolds, P. (1990), *SRADS with Local Rollback.* SCS Multi-Simulation Conference, Washington D.C., S. 161 – 164

[DIS80a] Dijkstra, E. und Scholten, C. (1980), *Termination Detection for Diffusing Computations.* Information Processing Letters, Vol. 11, Nr. 1, S. 1 – 4

[DMS98a] Defense Modeling and Simulation Office (1998), *DMSO Homepage.* http://www.dmso.mil/

[DMS99a] Defense Modeling and Simulation Office (1999), *DMSO High Level Architecture - Home.* http://hla.dmso.mil/hla/

[DUD91a] Dudenredaktion (1991), *Duden, Rechtschreibung der deutschen Sprache, Band 1.* Brockhaus-Verlag, Mannheim

[DUD96a] Dutt, S. und Deng, W. (1996), *A Probability-Based Approach to VLSI Circuit Partitioning.* Proc. of the 33th DAC, ACM/IEEE, S. 100 – 105, http://www.sigda.acm.org/Dac/33dac/papers/1996/dac96/pdffiles/06_4.pdf

[DUT93a] Dutt, S. (1993), *New Faster Kernighan-Lin-Type Graph-Partitioning Algorithms.* Int'l Conf. on Computer-Aided Design (ICCAD), S. 370 – 377

[FER96a] Ferscha, A. und Richter, M. (1996), *Massively Parallel Simulation of Business Process Models.* Proceedings of the ESM 1996, S. 377 – 381, http://sokrates.ani.univie.ac.at/ ferscha/E-PAPERS/esm96.ps.gz

[FET94a] Ferscha, A. und Tripathi, S. (1994), *Parallel and Distributed Simulation of Discrete Event Systems.* Technical Report, University of Maryland

[FGK97a] Formella, A., Grün, T., und Kessler, C. (1997), *The SB-PRAM: Concept, Design and Construction.* Proceedings of 3rd International Working Conference on Massively Parallel Programming Models (MPPM-97), http://www-wjp.cs.uni-sb.de/ lsbib/papers/PRAM/mppm.ps.gz

[FIM82b] Fiduccia, C. und Mattheises, R. (1982), *A Linear-Time Heuristic for Improving Network Partitions.* Proc. of the 19th DAC, ACM/IEEE, S. 175 – 181

[FIS95a] Fishwick, P. (1995), *Computer Simulation: The Art and Science of Digital World Construction.* http://www.cis.ufl.edu/ fishwick/introsim/paper.html

[FIS99a] Fishwick, P. (1999), *Web-Based Simulation Support.* http://www.cise.ufl.edu/ fishwick/websim.html

[FTG88a] Fujimoto, R., Tsai, J., und Gopalakrishnan, G. (1988), *Design and Evaluation of the Rollback Chip: Special Purpose Hardware for Time Warp.* Technical Report UUCS-88-011, Department of Computer Science, University of Utah

[FUJ88a] Fujimoto, R. (1988), *Performance Measurements of Distributed Simulation Strategies.* Proceedings of the Distributed Simulation Conference, San Diego, S. 14 – 20

[FUJ88b] Fujimoto, R. (1988), *Lookahead in Parallel Discrete Event Simulation.* Proceedings of the International Conference on Parallel Processing, S. 34 – 41

[FUJ89b] Fujimoto, R. (1989), *Time Warp on a Shared Memory Multiprocessor.* University of Utah, Salt Lake City, Technical Report UUCS-88-021a

[FUJ89d] Fujimoto, R. (1989), *Time Warp on a Shared Memory Multiprocessor.* Transactions of the Society for Computer Simulation, Vol. 6, Nr. 3, S. 211 – 239

[FUJ90a] Fujimoto, R. (1990), *Parallel Discrete Event Simulation.* Comm. of the ACM, Vol. 33, Nr. 10, S. 30 – 53

[FUJ93a] FUJIMOTO, R. (1993), *Parallel and Distributed Discrete Event Simulation: Algorithms and Applications.* Proceedings of the Winter Simulation Conference, S. 106 – 114

[FUJ93b] Fujimoto, R. M. (1993), *Parallel Discrete Event Simulation: Will the Field Survive?* ORSA Journal on Computing, Vol. 5, Nr. 3, S. 213 – 230

[FUJ93c] Fujimoto, R. M. (1993), *Future Directions in Parallel Simulation Research.* ORSA Journal on Computing, Vol. 5, Nr. 3, S. 245 – 248

[FUJ95a] FUJIMOTO, R. (1995), *Parallel and Distributed Simulation.* Proceedings of the Winter Simulation Conference, S. 118 – 125

[FUN92a] FUJIMOTO, R. und NICOL, D. (1992), *State of the Art in Parallel Simulation.* Proceedings of the Winter Simulation Conference, S. 246 – 254

[FWW84a] FRANKLIN, M., WANN, D., und WONG, K. (1984), *Parallel Machines and Algorithms for Discrete-Event Simulation.* Proceedings of the International Conference on Parallel Processing, S. 449 – 458

[GAF85b] GAFNI, A. (1985), *Space Management and Cancellation Mechanisms for Time Warp.* Ph.D. Dissertation, TR-85-341, Dept. of Computer Science, University of Southern California

[GAF88a] GAFNI, A. (1988), *Rollback mechanisms for Optimistic Distributed Simulation Systems.* Proc. of the SCS Multiconference on Distributed Simulation, Vol. 19, Nr. 3, S. 61 – 67

[GFU95a] GOMES, F., FRANKS, S., UNGER, B. W., ZHONGE, X., CLEARY, J. G., und COVINGTON, A. (1995), *SimKit: A High Performance Logical Process Simulation Class Library in C++.* Proceedings of the Winter Simulation Conference, S. 706 – 713, http://www.cpsc.ucalgary.ca/ gomes/PAPERS/SimKit_WSC95.ps

[GUN94a] GUNTER, M. (1991), *Supercritical Speedup.* 24th Annual Simulation Symposium, S. 159 – 168

[HAF85a] HAHN, W. und FISCHER, K. (1985), *An Event-Flow Computer for Fast Simulation of Digital Systems.* Proc. of the 22nd Design Automation Conference, ACM/IEEE, S. 338 – 344

[HEF91a] HEATH, M. und ETHERIDGE FINGER, J. (1991), *Visualizing Performance of Parallel Programms.* TR ORNL/TM-11813, Oak Ridge Nat. Lab.

[HEI86a] HEIDELBERGER, P. (1986), *Statistical Analysis of Parallel Simulations.* Proceedings of the Winter Simulation Conference, S. 290 – 295

[HEN96a] HEIDELBERGER, P. und D., N. (1996), *Parallel Execution for Serial Simulators.* ACM Transactions on Modeling and Computer Simulation, Vol. 6, Nr. 3, S. 210 – 242, http://www.cs.dartmouth.edu/ nicol/papers/ups.ps

[HIL85a] (1985), *HILO-3 User's Manual.* Fareham, England, http://www.semco.com/semcoate1.htm

[HOA78a] HOARE, C. (1978), *Communicating Sequential Processes.* Comm. of the ACM, Vol. 21, Nr. 8, S. 666 – 677

[HOC97a] HOOGSTRAETEN, W. v. und CORPORAAL, H. (1997), *ADVISE! Performance Evaluation of Parallel VHDL Simulation.* Proceedings of the 1997 Annual Simulation Symposium, S. 146 – 156, http://cardit.et.tudelft.nl/MOVE/papers/advise.ps

[HOC97b] HOCHBERGER, C. (1997), *The CEPRA-1X Cellular Processor.* Proc. of the Int. Parallel Processing Symposium

[HWB84a] Hwang, K. und Briggs, F. (1984), *Computer Architecture and Parallel Processing.* Computer Science Series. McGraw-Hill, New York

[IEE99a] IEEE (1999), *IEEE 1076.1 Analog and Mixed-Signal Extensions to VHDL.* http://vhdl.org/vi/analog/

[INT91a] Intel Corp. (1991), *Intel iPSC/2 and iPSC/860 User's Guide.* Intel Cooperation

[INT98a] Intusoft (1999), *XSPICE Overview.* http://www.intusoft.com/articles/xspiceover.htm

[JBW87a] Jefferson, D., Beckmann, B., Wieland, F., Blume, L., Diloreto, M., Hontalas, P., Laroche, P., Sturedevant, K., Typman, J., Vanarren, Wedel, J., Younger, H., und Bellenot, S. (1987), *Distributed Simulation and the Time Warp Operating System.* Proceedings of the 11th ACM Symposium on Operating Systems Principles, S. 77 – 93

[JEF85a] Jefferson, D. (1985), *Virtual Time.* ACM Transactions on Programming Languages and Systems, Vol. 7, Nr. 3, S. 404 – 425

[JEF90a] Jefferson, D. (1990), *Virtual Time II: Storage Management in Distributed Simulation.* Proc. of the Ninth Annual ACM Symposium on Principles of Distributed Computing, S. 75 – 89

[JER91a] Jefferson, D. und Reiher, P. (1994), *Understanding Supercritical Speedup.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 81 – 87

[JES85a] Jefferson, D. und Sowizral, H. (1985), *Fast Concurrent Simulation Using the Time Warp Mechanism.* Proceedings of the Conference on Distributed Simulation, S. 63 – 69

[JHB96a] Jha, V., Bagrodia, R. L., und Waldorf, J. (1996), *A Performance Evaluation Methodology for Parallel Simulation Protocols.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 180 – 183, ftp://may.cs.ucla.edu/pub/papers/pads96-perf.ps.gz

[JOD98a] Jones, K. und Das, S. R. (1998), *Combinig Optimism Limiting Schemes in Time Warp Based Parallel Simulation.* Proceedings of the Winter Simulation Conference, S. 290 – 295, http://ringer.cs.utsa.edu/šamir/Pubs/wsc98.ps

[JOH96a] Johannes, F. M. (1996), *Partitioning of VLSI Circuits and Systems.* Proc. of the 33th DAC, ACM/IEEE, S. 83 – 87, http://www.sigda.acm.org/Dac/33dac/papers/1996/dac96/pdffiles/06_1.pdf

[JQN92a] Johnson, B., Quarles, T., Newton, A., Pederson, D., und Sngiovanni-Vincentelli, A. (1992), *SPICE3 Version 3f User's Manual.* http://hera.eecs.berkeley.edu:80/ software/spice3f5.html

[KAA92a] Karthik, S. und Abraham, J. (1992), *Distributed VLSI Simulation on a Network of Workstations.* Int. Conference on Computer Desing: VLSI in Computers & Processors, S. 508 – 511

[KEL70a] Kernighan, B. und Lin, S. (1970), *An Efficient Heuristic Procedure for Partitioning Graphs.* Bell System Technical Journal, Vol. 49, S. 291 – 307

[KEL95a] Keller, A. (1995), *Internes Dokument: PVM/PARMACS Benchmarks.* http://www.uni-paderborn.de/pc2/systems/gcpp/doc/pvm_perf.pdf

[KGR94a] Kumar, V., Grama, A., und Vempaty, N. R. (1994), *Scalable Load Balancing Techniques for Parallel Computers.* Journal of Parallel and Distributed Computing, Vol. 22, S. 60 – 79, ftp://ftp.cs.umn.edu/users/kumar/lb_MIMD.ps

[KGV83a] Kirkpatrick, S., Gelatt, C., und Vecchi, M. (1983), *Optimization by Simulated Annealing.* Science, Vol. 220, S. 671 – 680

[KIJ96a] Kim, H. K. und Jean, J. (1996), *Concurrency Preserving Partitioning (CPP) for Parallel Logic Simulation.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 98 – 105

[KMC97a] Kormicki, M., Mahmood, A., und Carlson, B. S. (1997), *Parallel logic simulation on a network of workstations using parallel virtual machine.* ACM Transactions on Design Automation of Electronic Systems, Vol. 2, Nr. 2, S. 123 – 134, http://www.acm.org/pubs/citations/journals/todaes/1997-2-2/p123-kormicki/

[KMV83a] Kiviat, P., Markowitz, H., und Villanueva, R. (1983), *SIMSCRIPT II.5 Programming Language.* CACI, Los Angeles

[KOT94a] Koch, M. und Tavangarian, D. (1994), *Verteilte Simulation digitaler Systeme mit VHDL.* Proc. of ASIM Workshop

[KRA88a] Kravitz, S. und Ackland, B. (1988), *Static vs. dynamic partitioning of circuits for MOS timing simulator on a message-based multiprocessor.* In Unger, B. und Jefferson, D., Hrsg., *Distributed Simulation, 1988*, S. 136 – 140

[KRB96a] Krishnaswamy, V. und Banerjee, P. (1996), *Actor Based Parallel VHDL Simulation Using Time Warp.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 28, S. 135 – 142, http://www.acm.org/pubs/citations/proceedings/simulation/238788/p135-krishnaswamy/

[KSR94a] Riley, B. A. (1994), *Hardware Description of KSR1-64.*

[LAB92a] Lanchès, P. und Baitinger, U. (1992), *A Parallel Evaluation Environment for Distributed Logic Simulation.* In Stephenson, J., Hrsg., Modelling and Simulation, S. 465 – 469

[LAB92b] Lanchès, P. und Baitinger, U. (1992), *Parallele Vorverarbeitungsschritte für die verteilte Logiksimulation.* In Brauer, W., Hrsg., Parallele Datenverarbeitung mit dem Transputer, 4. Transputer-Anwender-Treffen TAT'92, S. 214 – 224, Springer-Verlag, Berlin, Heidelberg, NewYork, Tokyo

[LAM78a] Lamport, L. (1978), *Time, Clocks, and the Ordering of Events in a Distributed System.* Comm. of the ACM, Vol. 21, Nr. 7, S. 558 – 565

[LCU88a] Lomow, G., Cleary, J., Unger, B., und West, D. (1988), *A Performance Study of Time Warp.* Proceedings of the Distributed Simulation Conference, San Diego

[LEV93a] Levitan, S. (1993), *VCOMP & VSIM Reference Manual, Edition 1.2.1.*

[LIL89a] Lin, Y. und Lazowska, E. (1989), *The Optimal Checkpoint Interval in Time Warp Parallel Simulation.* Technical Report 89-09-04, University of Washington, Seattle

[LIL89b] Lin, Y. und Lazowska, E. (1989), *Optimality Considerations for Time Warp Parallel Simulation.* Technical Report 89-07-05, University of Washington, Seattle

[LIL89c] Lin, Y. und Lazowska, E. (1989), *A Study of Time Warp Rollback Mechanisms.* Technical Report 89-09-07, University of Washington, Seattle

[LIL89d] Lin, Y. und Lazowska, E. (1989), *Exploiting Lookahead in Parallel Simulation.* Technical Report 89-10-06, University of Washington, Seattle

[LIL90a] Lin, Y. und Lazowska, E. (1990), *Reducing the State Saving Overhead for Time Warp Parallel Simulation.* Proc. International Conference on Parallel Processing, Vol. 3, S. 201 – 209

[LIL90b] Lin, Y. und Lazowska, E. (1990), *Processor Scheduling for Time Warp Parallel Simulation.* Technical Report 90-03-03, University of Washington, Seattle

[LIL90c] Lin, Y. und Lazowska, E. (1990), *Optimiality Considerations of Time Warp Parallel Simulation.* Distributed Simulation

[LIN92a] Lin, Y.-B. (1992), *Parallelism Analyzers for Parallel Discrete Event Simulation.* ACM Transactions on Modelling and Computer Simulation, Vol. 2, Nr. 3, S. 239 – 264

[LIN93a] Lin, Y.-B. (1993), *Will Parallel Simulation Research Survive?* ORSA Journal on Computing, Vol. 5, Nr. 3, S. 236 – 238

[LLB89a] Lin, Y., Lazowska, E., und Bailey, M. (1989), *Comparing Synchronization Protocols for Parallel Logic-Level Simulation.* Technical Report 89-09-06, University of Washington, Seattle, Washington, Dept. of Computer Science and Engineering

[LLB89b] Lin, Y., Lazowska, E., und Baer, J. (1989), *Conservative Parallel Simulation for Systems with no Lookahead Prediction.* Technical Report 89-07-07, University of Washington, Seattle

[LLM87a] Litzkow, M., Livny, M., und Mutka, M. (1987), *CONDOR - A Hunter of idle Workstations.* Technical Report 730, University of Wisconsin, Dept. of Computer Science, Madison

[LMP82a] Levendel, Y., Menon, P., und Patel, S. (1982), *Special Purpose Computer for Logic Simulation Using Distributed Processing.* Bell System Technical Journal, Vol. 61, Nr. 10, S. 2873 – 2909

[LPL93a] Lin, Y.-B., Preiss, B. R., Loucks, W. M., und Lazowska, E. D. (1993), *Selecting the Checkpoint Interval in Time Warp Simulation.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 3 – 10

[LSW89a] Lubachevsky, B., Shwartz, A., und Weiss, A. (1989), *Rollback Sometimes Works ... If Filtered.* Proceedings of the Winter Simulation Conference

[LUB89a] Lubachevsky, B. (1989), *Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks.* Comm. of the ACM, S. 111 – 123

[LUK93a] Luksch, P. (1993), *Parallelisierung ereignisgetriebener Simulationsverfahren auf Mehrprozesoorsystemen mit verteiltem Speicher.* Verlag Dr. Kovač, Hamburg

[LUK95a] Luksch, P. (1995), *Parallel Logic Simulation on Distributed Memory Multiprocessors – Classification and Evaluation of Different Approaches.* Systems Analysis Modelling Simulation, Vol. 21, S. 123 – 145, http://wwwbode.informatik.tu-muenchen.de/archiv/artikel/SAMS/sams.ps.gz

[MAL93a] Manjikian, N. und Loucks, W. (1993), *High Performance Parallel Logic Simulation on a Network of Workstations.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 76 – 84

[MaM89a] Mattern, F. und Mehl, H. (1989), *Diskrete Simulation – Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung.* Informatik-Spektrum, Vol. 12, Nr. 4, S. 198 – 210

[MAR95a] Marks, M. (1995), **IN***vestigation through* **V***isualization and* **A***nalysis of* **D***iscrete* **E***vent* **S***imulation Traces (INVADES): Implementierung graphischer Darstellungsmöglichkeiten – Bewertung des Logiksimulators DVSIM durch INVADES.* Diplomarbeit, Universität des Saarlandes, Saarbrücken

[MAT87a] Mattern, F. (1987), *Algorithms for Distributed Termination Detection.* Distributed Computing 2, S. 161 – 175

[MAT89d] Mattern, F. (1989), *Verteilte Basisalgorithmen.* Springer-Verlag, Informatik-Fachberichte Bd. 226

[MAT89f] Mattern, F. (1989), *Global Quiescence Detection Based on Credit Distribution and Recovery.* Information Processing Letters 30, S. 195 – 200

[MAT92a] Matsumoto, Y. und Taki, K. (1992), *Parallel Logic Simulation on a Distributed Memory Machine.* Proc. of the European Conference on Design Automation, S. 76 – 80

[MAT93a] Mattern, F. (1993), *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation.* Journal of Parallel and Distributed Simulation, Vol. 18, S. 423 – 434

[MBW96a] Martin, D., McBrayer, T., und Wilsey, P. A. (1996), *warped: A Time Warp Simulation Kernel for Analysis and Application Development.* Proc. of the 29th Hawaii Intl. Conf. on System Sciences (IEEE), Vol. 29, Nr. 1, S. 383 – 386

[MEH84a] Mehlhorn, K. (1984), *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness.* Springer-Verlag, Berlin, Heidelberg, NewYork, Tokyo

[MEH91a] Mehl, H. (1991), *Speed-Up of Conservative Distributed Discrete Event Simulation Methods by Speculative Computing.* Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation, Anaheim, California, Simulation Series, Hrsg.: V. Madisetti, D. Nicol, R. Fujimoto, Vol. 23, Nr. 1, S. 163 – 166

[MEH91b] Mehl, H. (1991), *DSL –Eine verteilte Simulationssprache.* Interner Arbeitsbericht, Fachbereich Informatik, Universität Kaiserslautern

[MEH91c] Mehl, H. (1991), *Breaking Ties Deterministically in Distributed Simulation Schemes.* Technischer Bericht SFB 124-217/91, Fachbereich Informatik, Universität Kaiserslautern

[MEH93a] Mehl, H. und Hammes, S. (1993), *Shared Variables in Distributed Simulation.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 68 – 75, http://www.acm.org/pubs/citations/proceedings/simulation/158459/p68-mehl/

[MEH94a] Mehl, H. (1994), *Methoden verteilter Simulation.* Vieweg-Verlag, Braunschweig

[MEI91a] Meister, G. (1991), *Verteilte Simulation von Warteschlangennetzen auf Transputern.* http://www.informatik.tu-darmstadt.de/VS/Mitarbeiter/Meister/publications/projekt.ps

[MEI91b] Meister, G. (1991), *Extension to DISQUE - A trace facility to produce trace data for use by a monitoring tool for distributed simulators.* http://www.informatik.tu-darmstadt.de/VS/Mitarbeiter/Meister/publications/monitor.ps

[MEI93a] Meister, G. (1993), *A Survey on Parallel Logic Simulation.* Technischer Bericht SFB 124, TR-14/1993, Fachbereich Informatik, Universität des Saarlandes, 1993

[MEI96a] Meister, G. (1996), *Evaluation of Parallel Logic Simulation Using DVSIM.* Proc. of the 29th Hawaii Intl. Conf. on System Sciences (IEEE), Vol. 29, Nr. 1, S. 397 – 406

[MHF92a] Madisetti, V., Hardaker, D., und Fujimoto, R. (1992), *The Mimdix Operating System for Parallel Simulation.* Proc. of the 6th Workshop on Parallel and Distributed Simulation (PADS), Vol. 24, S. 65 – 74

[MIS86a] Misra, J. (1986), *Distributed Discrete-Event Simulation.* Computing Surveys, Vol. 18, Nr. 1, S. 39 – 65

[MIT99a] The Mitre Corp. (1999), *Web-Based Simulation Support.* http://ms.ie.org/websim/

[MMR93a] Mattern, F., Mehl, H., und Richter, J. (1993), *Leistungssteigerung ereignisgesteuerter Simulation durch Multi-Mikro-Rechnersysteme.* Abschlußbericht zum gleichnamigen DFG-Projekt, Universität des Saarlandes

[MMS91a] Mattern, F., Mehl, H., Schoone, A., und Tel, G. (1991) , *Global Virtual Time Approximation with Distributed Termination Detection Algorithms.* Technical Report RUU –CS – 91-32, University of Utrecht

[MPI93a] The MPI Forum (1993), *MPI: A Message Passing Interface.* Proceedings of Supercomputing 1993

[MSD89a] Müller-Thuns, R., Saab, D., Damiano, R., und Abraham, J. (1989), *Portable Parallel Logic and Fault Simulation.* Proc. International Conference on Computer-Aided Design, 1989, S. 506 – 509

[MWM88a] Madisetti, V., Walrand, J., und Messerschmitt, D. (1988), *WOLF: A Rollback Algorithm for Optimistic Distributed Simulation Systems.* Proceedings of the Winter Simulation Conference, S. 296 – 305

[NAL93a] Nandy, B. und Loucks, W. (1993), *On a Parallel Partitioning Technique for Use with Conservative Parallel Simulation.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 43 – 51

[NIF94a] Nicol, D. und FUJIMOTO, R. (1994), *Parallel Simulation Today.* Operations Research

[NIH96a] Nicol, D. und Heidelberger, P. (1996), *On Extending More Parallelism to Serial Simulators.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 28, S. 202 – 205

[NMI90a] Nicol, D., Micheal, C., und Inouye, P. (1990), *Efficient Aggregation of Multiple LPs in Distributed Memory Parallel Simulation.*

[ORC99a] Orcad Incorp. (1999), *PSpice Technical Services.* http://www.orcad.com/techserv/spicesupp_f.htm

[ORS93a] Fujimoto , R. M. et al. (1993), *Parallel Discrete Event Simulation.* ORSA Journal on Computing, Vol. 5, Nr. 3

[PAR93a] Pancerella, C. und Reynolds, Jr., P. (1993), *Disseminating Critical Target-Specific Synchronization Information in Parallel Discrete Event Simulations.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 52 – 59

[PAR94a] Parsytec GmbH (1994), *PowerPVM: Design and Implementation Description.* Aachen, http://www.uni-paderborn.de/pc2/software/ppvm11/docus/design.pdf

[PAR95a] Parsytec GmbH (1995), *Parix Version 1.3.1-PPC Software Documentation.* Aachen, http://www.uni-paderborn.de/pc2/software/parix131/docu/parix131.ps.Z

[PAW92a] Palaniswamy, A., Aji, S., und Wilsey, P. (1992), *An Efficient Implementation of Lazy Reevaluation.* Proc. of the 25th Annual Simulation Symposium

[PAW93a] Palaniswamy, A. C. und Wilsey, P. A. (1993), *An Analytical Comparison of Periodic Checkpointing and Incremental State Saving.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 127 – 134

[PED96a] Peterson, Larry, L. und Davie, B. S. (1996), *Computer Networks – A Systems Approach.* Morgan Kaufmann Publishers, Inc., San Francisco

[PLH88a] Preiss, B., Loucks, W., und Hamacher, V. (1988), *A Unified Modeling Methodology for Performance Evaluation of Distributed Discrete Event Simulation Mechanism.* Proceedings of the Winter Simulation Conference, S. 315 – 324

[PPC96a] Parsytec GmbH (1996), *PPC User's Guide*. Aachen, Technischer Bericht

[PRM90a] Preiss, B. und Macintyre, I. (1990), *YADDES - Yet Another Distributed Discrete Event Simulator: User Manual*. Technical Report, CCNG E-197, University of Waterloo, Waterloo

[PRW99a] Preiss, B. R. und Wan, C. K. W. (1999), *The Parsimony Project: A Distributed Simulation Testbed in Java*. Proc. Intl. Conf. On Web-Based Modelling & Simulation, S. 6 ff., http://dictator.uwaterloo.ca/Bruno.Preiss/papers/published/1999/websim/

[PUG90a] Pugh, W. (1990), *Skip Lists: A Probabilistic Alternative to Balanced Trees*. Comm. of the ACM, Vol. 33, Nr. 6, S. 668 – 676, http://www.acm.org/pubs/citations/journals/cacm/1990-33-6/p668-pugh/

[PWM79a] Peacock, K., Wong, J., und Manning, E. (1979), *Distributed Simulation Using a Network of Processors*. Computer Networks 3, S. 44 – 56

[RAT93a] Rajaei, H., Ayani, R., und Thorelli, L. (1993), *The Local Time Warp Approach to Parallel Simulation*. Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 119 – 126

[RED89a] Reynolds, P. und Dickens, P. (1989), *SPECTRUM: A Parallel Simulation Testbed*. 4th Annual Hypercube Conference, Monterey

[REI92a] Reiher, P. L. (1992), *Experiences with Optimistic Synchronization for Distributed Operating Systems*. 3rd Symposium on Experiments with Distributed and Multiprocessor Systems, ftp://ftp.cs.ucla.edu/pub/ficus/reiher/tw.papers/tw_experience.ps.gz

[REJ90a] Reiher, P. und Jefferson, D. (1990), *Dynamic Load Management in the Time Warp Operating System*. Transactions of the Society for Computer Simulation, Vol. 7, Nr. 2, S. 91 – 120

[REY82a] Reynolds, P. (1982), *A Shared Resource Algorithm for Distributed Simulation*. Proceedings of the 9th IEEE Symposium on Computer Architecture, S. 259 – 266

[REY88a] Reynolds, P. (1988), *A Spectrum of Options for Parallel Simulation*. Proceedings of the Winter Simulation Conference, S. 325 – 332

[REY93a] Reynolds Jr., P. F. (1993), *The Silver Bullet*. ORSA Journal on Computing, Vol. 5, Nr. 3, S. 239 – 241

[RFB90a] Reiher, P., Fujimoto, R., Bellenot, S., und Jefferson, D. (1990), *Cancellation Strategies in Optimistic Execution Systems*. Proc. of the SCS Multiconference on Distributed Simulation, Vol. 22, S. 112 – 121

[RIC91a] Richter, J. (1991), *DISQUE: Ein verteilter Simulator für Warteschlangennetze auf Transputern*. In Grebe, R. und Baumann, M., Hrsg., Parallele Datenverarbeitung mit dem Transputer. 3. Transputer-Anwender-Treffen TAT'91, Informatik aktuell, Berlin, Heidelberg, NewYork, Tokio, Springer-Verlag, S. 345 – 352

[RIC95a] Richter, J. (1995), *Perspektiven der parallelen, ereignisgesteuerten Simulation am Beispiel von Warteschlangennetzen*. Dissertation, Fachbereich Informatik, Universität des Saarlandes

[RIW89a] Righter, R. und Walrand, J. (1989), *Distributed Simulation of Discrete Event Systems.* Proceedings of the IEEE,, Vol. 77, Nr. 1, S. 99 – 113

[RLA96a] Rönngren, R., Liljenstam, M., Ayani, R., und Montagnat, J. (1996), *Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 70 – 77

[RMM88a] Reed, D., Malony, A., und Mccredie, B. (1988), *Parallel Discrete Event Simulation Using Shared Memory.* IEEE Transactions on Software Engineering, Vol. 14, Nr. 4, S. 541 – 553

[ROS92a] Röhm, E. und Schütz, M. (1992), *Hardware-Beschreibungssprache VHDL.* Schulungsunterlagen, Deutsche Informatik Akademie, Bonn

[RUN88a] Runo, D. (1988), *Das Graphpartitionierungsproblem und seine Lösungsmethoden.* Diplomarbeit, GMD, Bonn

[RWF89a] Reynolds Jr., P., Weight, C., und Fidler II, J. (1989), *Comparative Analyses of Parallel Simulation Protocols.* Proceedings of the Winter Simulation Conference

[RWH90a] Reiher, L., Wieland, F., und Hontalas, P. (1990), *Providing Determinism in the Time Warp Operating System – Costs, Benefits, and Implications.* Proceedings of the Second IEEE Workshop on Experimental Distributed Systems, Huntsville, Alabama, S. 113 – 118

[RWH92a] Reiher, P. L., Wieland, F., und Hontales, P. (1990), *Providing Determinism in the Time Warp Operating System – Costs Benefits, and Implications.* IEEE Workshop on Experimental Distributed Systems, ftp://ftp.cs.ucla.edu/pub/ficus/reiher/tw.papers/tw_determ.ps.gz

[SAM85a] Samadi, B. (1985), *Distributed Simulation: Performance and Analysis.* Ph.D. thesis, Dept. of Computer Science, University of California at Los Angeles

[SAS90a] Sang, J. und Sang, M. (1990), *Untersuchung von Algorithmen zur verteilten ereignisgesteuerten Simulation am Beispiel eines parallelen Multi-Level-Simulators auf Transputerbasis.* Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl I

[SBW88a] Sokol, L., Briscoe, D., und Wieland, A. (1988), *MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution.* Proceedings of the Distributed Simulation Conference, San Diego, S. 1 – 21

[SCH74a] Schriber, T. (1974), *Simulation Using GPSS.* John Wiley & Sons

[SCH86b] Schwetman, H. (1986), *A C-based, Process-Oriented Simulation Language.* Proceedings of the Winter Simulation Conference, S. 387 – 396

[SEQ99a] Sequent Computer Systems (1999), *Sequent - Symmetry Overview.* http://www.sequent.com

[SER92a] Setz, T. und Roth, R. (1992), *LIPS: a System for Distributed Processing on Workstations.* Technischer Bericht SFB 124 TP D5, Universität des Saarlandes, Saarbrücken

[SES85a] Sechen, C. und Sangiovanni-Vincentelli, A. (1985), *The TimberWolf Placement and Routing Package.* IEEE Journal of Solid-State Circuits, Vol. SC-20, Nr. 2, S. 510 – 522

[SGD93a] Sunderam, V., Geist, G., Dongarra, J., und Manchek, R. (1993?), *The PVM Concurrent Computing System: Evolution, Experiences, and Trends.* Journal of Parallel Computing

[SIM94a] Simmet, F.-J. (1994), *Entwurf und Implementierung eines verteilten VHDL-Simulators, Teil 1.* Diplomarbeit, Universität des Saarlandes, Saarbrücken, http://www.informatik.tu-darmstadt.de/VS/Mitarbeiter/Meister/project/dvsim/DVSIM_DOKU_FJ.ps.gz

[SIM94b] Simon, J. (1994), *Leistung eines Parallelrechners auf Basis des PowerPC-Prozessors.* Parallele Datenverarbeitung aktuell: TAT 94 - Parallelrechner Grundlagen und Anwendungen, http://www.uni-paderborn.de/pc2/services/public/1994/94004.pdf

[SIO93a] Simic, N. und Ortner, H. (1993), *Partitioning Strategies within a Distributed Multilevel Logic Simulator including Dynamic Repartitioning.* Proc. of the European Design Automation Conference

[SIS94a] Singhal, M. und Shivaratri, N. G. (1994), *Advanced Concepts in Operating Systems.* McGraw-Hill, New York, NY

[SIS99a] Simulation Interoperability Standards Organization (1999), *DIS Material.* http://siso.sc.ist.ucf.edu/dis/index.htm

[SKP93a] Shanker, M. S., Kelton, W. D., und Padman, R. (1993), *Measuring Congestion for Dynamic Task Allocation in Distributed Simulation.* ORSA Journal on Computing, Vol. 5, Nr. 1, S. 54 – 68

[SKS92a] Shivaratri, N. G., Krueger, P., und Singhal, M. (1992), *Load Distributing for Locally Distributed Systems.* IEEE Computer, S. 33 – 44

[SMI86a] Smith, II, R. (1986), *Fundamentals of Parallel Logic Simulation.* Proc. of the 23th DAC, ACM/IEEE, S. 2 – 12

[SMU87a] Smith, S., Mercer, M., und Underwood, B. (1987), *An Analysis of Several Approaches to Circuit Partitioning for Parallel Logic Simulation.* Proc. of Int. Conference on Computer Design, IEEE, S. 664 – 667

[SOU92a] Soulé, L. (1992), *Parallel Logic Simulation: An Evaluation of Centralized-Time and Distributed-Time Algorithms.* Ph.D. thesis, Stanford University, TR CSL-TR-92-527

[SP298a] Intel Corp. (1998), *IBM RS/6000: Hardware: Large Scale Server.* http://www.rs6000.ibm.com/hardware/largescale/index.html

[SRI95a] Srinivasan, R. (1995), *XDR: External Data Representation Standard, IETF, RFC 1832.* Sun Microsystems Inc., http://www.ietf.org/rfc/rfc1832.txt

[SRR95a] Srinivasan, S. und Reynolds Jr., P. F. (1995), *Adaptive Algorithms vs. Time Warp: An Analytical Comparison.* Proceedings of the 1995 Winter Simulation Conference, S. 666 – 673

[SRS95a] Schlagenhaft, R., Ruhwandl, M., Sporrer, C., und Bauer, H. (1995), *Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations.* Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS), Vol. 25, S. 175 – 180, http://www.acm.org/pubs/contents/proceedings/simulation/214282/index.html

[STE91a] Steinman, J. (1991), *Speedes: Synchronous Parallel Environment for Emulation and Discrete Event Simulation.* Advances in Parallel and Distributed Simulation, Vol. 23, S. 95 – 103

[STE93a] Steinman, J. (1993), *Breathing Time Warp.* Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS), Vol. 23, S. 109 – 118

[STE94a] Steinman, J. (1994), *Discrete Event Simulation and the Event Horizon.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), Vol. 24, S. 39 – 49

[STU93a] Sturm, B. (1993), *YES - Ein Werkzeug zur Analyse verteilter Simulationsalgorithmen.* Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern

[SUN90a] Sunderam, V. (1990), *PVM: A Framework for Parallel Distributed Computing.* Journal of Concurrency: Practice and Experience, Vol. 2, Nr. 4, S. 315 – 339

[SUS88a] Su, W.-K. und Seitz, C. L. (1988), *Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm.* Technical Report Caltech-CS-TR-88-22, California Institute of Technology

[SUS89a] Su, W.-K. und Seitz, C. (1989), *Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm.* Proceedings of the SCS Multiconference on Distributed Simulation, S. 38 – 43

[SWF87a] Swope, S. und Fujimoto, R. (1987), *Optimal Performance of Distributed Simulation Programs.* Proceedings of the Winter Simulation Conference, S. 612 – 617

[UCB97a] University of California, Berkeley (1997), *The Spice Home Page.* http://infopad.eecs.berkeley.edu/ icdesign/SPICE/

[UCC93a] Unger, B. W., Cleary, J. G., Covington, A., und Darrin, W. (1993), *An External State Management System for Optimistic Parallel Simulation.* Proceedings of the 1993 Winter Simulation Conference, S. 750 – 755

[UNC93a] Unger, B. W. und Cleary, J. G. (1993), *Practical Parallel Discrete Event Simulation.* ORSA Journal on Computing, Vol. 5, Nr. 3, S. 242 – 244

[VCW94a] Varghese, G., Chamberlain, R., und Weihl, W. (1994), *The pessimism behind optimistic simulation.* Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 126 – 131

[VEL92a] Vellandi, B. und Lightner, M. (1992), *Parallelism extraction and Program Restructuring of VHDL for Parallel Simulation.* Proc. of the EDAC

[VHD87a] IEEE. *IEEE Standard VHDL Language Reference Manual.* NY 10017, USA, IEEE STD. 1076 – 1987

[WAG95a] Walker, P. A. und Ghosh, S. (1995), *Asynchronous Distributed Event Driven Simualtion Algorithm for Execution of VHDL on Parallel Processors.* Proc. of the 32th DAC, ACM/IEEE, S. 144 – 150, http://www.acm.org/pubs/citations/proceedings/dac/217474/p144-walker/

[WEI94a] Weissenfels, J. (1994), *Entwurf und Implementierung eines verteilten VHDL-Simulators, Teil 2 - Die Simulatoren.* Diplomarbeit, Universität des Saarlandes, Saarbrücken, http://www.informatik.tu-darmstadt.de/VS/Mitarbeiter/Meister/project/dvsim/DVSIM_DOKU_JW.ps.gz

[WEM88a] West, J. und Mullarney, A. (1988), *ModSim: A Language for Distributed Simulation.* SCS Multiconference on Distributed Simulation, Vol. 20, S. 155 – 159

[WEP96a] West, D. und Panesar, K. (1996), *Automatic Incremental State Saving.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 26, S. 78 – 85

[WES88a] West, D. (1988), *Optimizing Time Warp: Lazy Rollback and Lazy Re-evaluation.* Master's thesis, University of Calgary

[WFC86a] Wong, K., Franklin, M., Chamberlain, R., und Shing, B. (1986), *Statistics on Logic Simulation.* Proc. of the 23rd DAC, ACM/IEEE, S. 13 – 19

[WHF89a] Wieland, F., Hayley, L., Feinberg, A., Diloreto, M., Blume, L., Ruffles, J., Reiher, P., Beckmann, B., Hontalas, P., Bellenot, S., und Jefferson, D. (1989), *The Performance of a Distributed Combat Simulation with the Time Warp Operating System.* Concurrency: Practice and Experience, Vol. 1, Nr. 1, S. 35 – 50

[WIJ89a] Wieland, F. und Jefferson, D. (1989), *Case Studies in Serial and Parallel Simulation.* International Conference on Parallel Processing, S. 255 – 258

[WIN96a] Wilson, L. und Nicol, D. (1996), *Experiments in Automated Load Balancing.* Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS), Vol. 28, S. 4 – 11

[WLB88a] Wagner, D., Lazowska, E., und Bershad, B. (1988), *Techniques for Efficient Shared Memory Parallel Simulation.* Technical Report 88-04-05, Department of Computer Science, University of Washington, Seattle

[WMS98a] Wilsey, P. A., Martin, D. E., und Subramani, K. (1998), *SAVANT/TyVIS/WARPED: Components for the Analysis and Simulation of VHDL.* VHDL Users' Group Spring 1998 Conference, S. 195 – 201, http://www.ece.uc.edu/ paw/lab/papers/savant/viuf-sp98.ps.gz

[WUZ98a] Williamson, C., Unger, B. W., und Zhonge, X. (1998), *Parallel Simulation of ATM Networks: Case Study and Lessons Learned.* Technical Report, University of Calgary, http://bungee.cpsc.ucalgary.ca/Publications/lessons.ps

Lebenslauf

Gerd Meister

geboren am 15. August 1964 in Kaiserslautern

1970 - 1974	Grundschule Mehlingen
1974 - 1983	Albert-Schweitzer-Gymnasium Kaiserslautern Abschluß: Abitur, 06/1983
1983 - 1985	Ausbildung zum Bau- u. Möbelschreiner Abschluß: Facharbeiterbrief, 07/1985
1985 - 1986	tätig als Schreiner
10/1986 - 03/1992	Studium der Informatik, Universität Kaiserslautern Abschluß: Diplom, 03/1992
04/1992 - 10/1994	Wissenschaftlicher Mitarbeiter im Fachbereich Informatik (SFB 124) der Universität des Saarlandes, Saarbrücken
10/1994 - 06/1999	Wissenschaftlicher Mitarbeiter im Fachbereich Informatik der Technischen Universität Darmstadt
seit 07/1992	Fachreferent bei Bosch Telecom GmbH, Frankfurt a. M.